RL-TR-96-3
In-House Report
February 1996

WAVES-VHDL INTEGRATION FOR COMMON APPLICATIONS

Steven Drager, Christopher Flynn, Frederick Hall,
James Hanna, Robert Hillman, and James Nagy

Rome Laboratory
Air Force Materiel Command
Rome, New York

RL-TR-96-3 has been reviewed and is approved for publication.

APPROVED: Eugene C. Blackburn

EUGENE C. BLACKBURN
Chief, Electronics Reliability Division
Electromagnetics & Reliability Directorate

FOR THE COMMANDER: John J. Bart

JOHN J. BART
Chief Scientist, Reliability Services
Electromagnetics & Reliability Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** February 1996	**3. REPORT TYPE AND DATES COVERED** In-House April 1995 - October 1995

4. TITLE AND SUBTITLE

WAVES-VHDL INTEGRATION FOR COMMON APPLICATIONS

5. FUNDING NUMBERS

PE - 62702F
PR - 2338
TA - 01
WU - 8P

6. AUTHOR(S)

Steven Drager, Christopher Flynn, Frederick Hall, James Hanna, Robert Hillman, and James Nagy

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Rome Laboratory (ERDD)
525 Brooks Rd.
Rome, NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER

RL-TR-96-3

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory (ERDD)
525 Brooks Rd.
Rome, NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Steven Drager/ERDD, (315)330-2735

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release, distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report provides an explanation of how to use the IEEE Waveform and Vector Exchange Specification (WAVES) language, an industry standard for representing digital stimulus and response data for both the design and test communities, in conjunction with the VHSIC Hardware Description Language. The common complaint heard through the user community has been the complexity and difficulty in using WAVES to verify a VHDL model. This work has addressed this complaint by creating four common library packages WAVES_1164_Pin_Codes, WAVES_1164_Logic_Value, WAVES_1164_Frames and WAVES_1164_Utilities as well as an automatic testbench generation tool to support the WAVES-VHDL testbench configuration and mitigate the complexity for the average user. The report first provides a discussion on each of these packages, so that the user may understand how the packages may be utilized. Then design examples are provided to demonstrate the usage and utility of the libraries as well as to walk the user through the usage of the testbench generation tool.

14. SUBJECT TERMS

VHSIC Hardware Description Language (VHDL), Waveform and Vector Exchange (WAVES)

15. NUMBER OF PAGES

128

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF FIGURES

TABLE OF LISTINGS

TABLE OF TABLES

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION

The intent of this Technical Report is to provide potential users with an explanation of how to use the IEEE Standard for Waveform and Vector Exchange (WAVES) [1] in conjunction with the VHSIC Hardware Description Language (VHDL). WAVES is an industry standard for representing digital stimulus and response data for both the design and test communities. The WAVES standard format was developed to support users in the exchange of waveform information between multiple simulator and tester environments. The WAVES standard provides for the faithful exchange of test verification data between systems. WAVES, being a non-proprietary standard, provides a design to test link between these systems by establishing a common data exchange capability.

This report intends to provide the reader with a basic understanding of how to utilize a common set of WAVES elements in VHDL design verification. The authors will first introduce the reader to a common set of support utilities for WAVES and relate them to a functional understanding of how they are utilized. This approach suggests a methodology that minimizes the work involved by an individual through the use of a common user library. This methodology does not restrict the user, but rather provides an organized scheme to eliminate repetitive work and reduce the learning cycle for WAVES.

The material presented in this report assumes that the user has a basic knowledge of VHDL modeling and test verification. The material addresses the needs of both the design engineer and the test engineer. For the designer, some familiarity or knowledge of VHDL is required in understanding the use of WAVES in conjunction with VHDL. The test engineer should be familiar with a high level programming language in order to understand how to represent a waveform for application on an Automated Test System (ATS).

1.2 WAVES History

The name WAVES was chosen carefully. The words "waveform" and "vector" indicate that WAVES may represent simulator event trace data, as well as the highly structured test vectors typical of automated test equipment. The word "exchange" means that WAVES is meant for the exchange of information between vendors as well as design and test environments. WAVES was not, however, designed to replace the stimulus/response formats used within a given environment.

Because WAVES is an exchange specification, all facets of the stimulus and response data must be captured. Nothing must be left to the reader's imagination, for everyone has slightly different interpretations about what constitutes expected data. What is "obviously right" to one engineer is likely to be "obviously wrong" to another. When exchanging information between environments, assumptions are dangerous and often incorrect. Therefore, WAVES data stands alone and does not require anything in common between sender and receiver, other than an adherence to the WAVES specification.

WAVES is a subset of IEEE Std 1076-1993, also known as VHDL [2]. Anyone with an understanding of VHDL will have an easy time understanding the syntax and semantics of WAVES. However, complete knowledge of VHDL is certainly not a prerequisite for understanding WAVES. WAVES includes only the sequential portion of VHDL, which means that WAVES is essentially an algorithmic programming language. Anyone familiar with a modern programming language, such as Ada or C++, will have no trouble reading WAVES source code.

VHDL was chosen as the basis for WAVES because VHDL is so important in the design phase of electronic components and modules. VHDL is a standard representation for modeling and simulating digital circuits. Extensive effort has gone into the design of VHDL, so there was no reason to repeat that work by designing a totally new language. Finally, many design environments are available for VHDL, and these may be used without change on a WAVES data set.

1.3 WAVES-VHDL Modeling and Simulation

The acceptance of a standard as a common practice is slow and requires an iterative process involving user demand, support tooling and user acceptability. Tool vendors will only invest in a tool development when it is apparent there is a profitable market available. Users, on the other hand, require both tool support as well as an understanding of the standard and the benefits of using that standard. The main ingredient to the adoption of a standard as a common practice is the availability of information for the use and application of the standard. The focus of this section is to discuss the work performed to enhance the integration of WAVES for model simulation and verification.

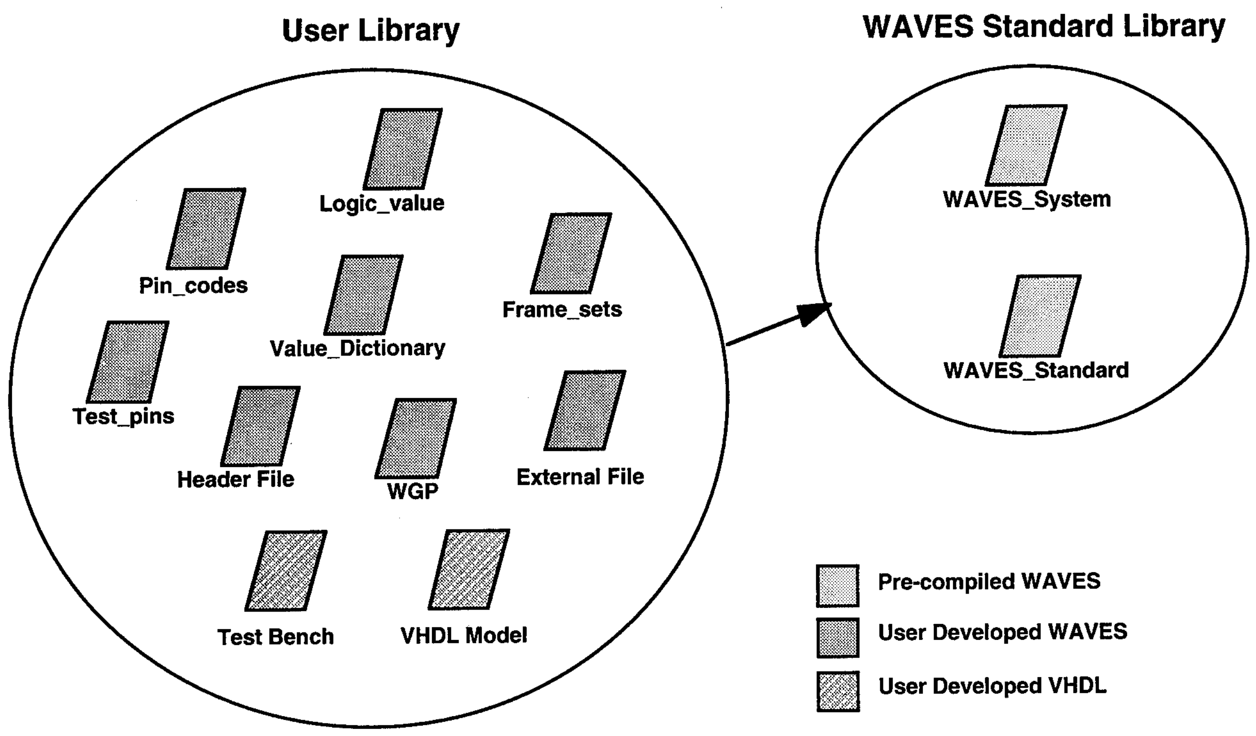


Figure 1.3.1 WAVES usage environment prior to this work.

The initial complexity of WAVES, prior to performing this work, is shown in Figure 1.3.1. The lack of tool support easing this complexity was identified as a weak point in achieving widespread WAVES utilization. In order to utilize WAVES, a person had to be well versed in the syntax, semantics and usage of WAVES so that they could create all of the necessary files to perform the simulation. Therefore, tools and libraries have been developed to mitigate the complexity for the average user.

Figure 1.3.2 shows the new library structure. The complexity of the User Library has been reduced through the creation of the WAVES 1164 Library, which provides a pre-defined environment for common applications of WAVES and VHDL. The WAVES 1164 Library elements were developed to reduce the learning curve by providing a set of predefined library functions which will meet 90% of all users needs. The hope is that these packages will eventually be integrated into the WAVES Language Reference Manual (LRM). Chapters 2, 3 and 4 will discuss each of the new functions placed in the WAVES 1164 Library.

A prototype automatic testbench generation tool has also been developed based on these library elements. The purpose of the tool is to provide a methodology that minimizes the work involved by an individual by exploiting the use of a common user library. The tool automatically generates a testbench, which not only applies WAVES stimulus to the model, but verifies the validity of the expected responses. Additionally, others tools have been developed to support the generation of the WAVES data set.

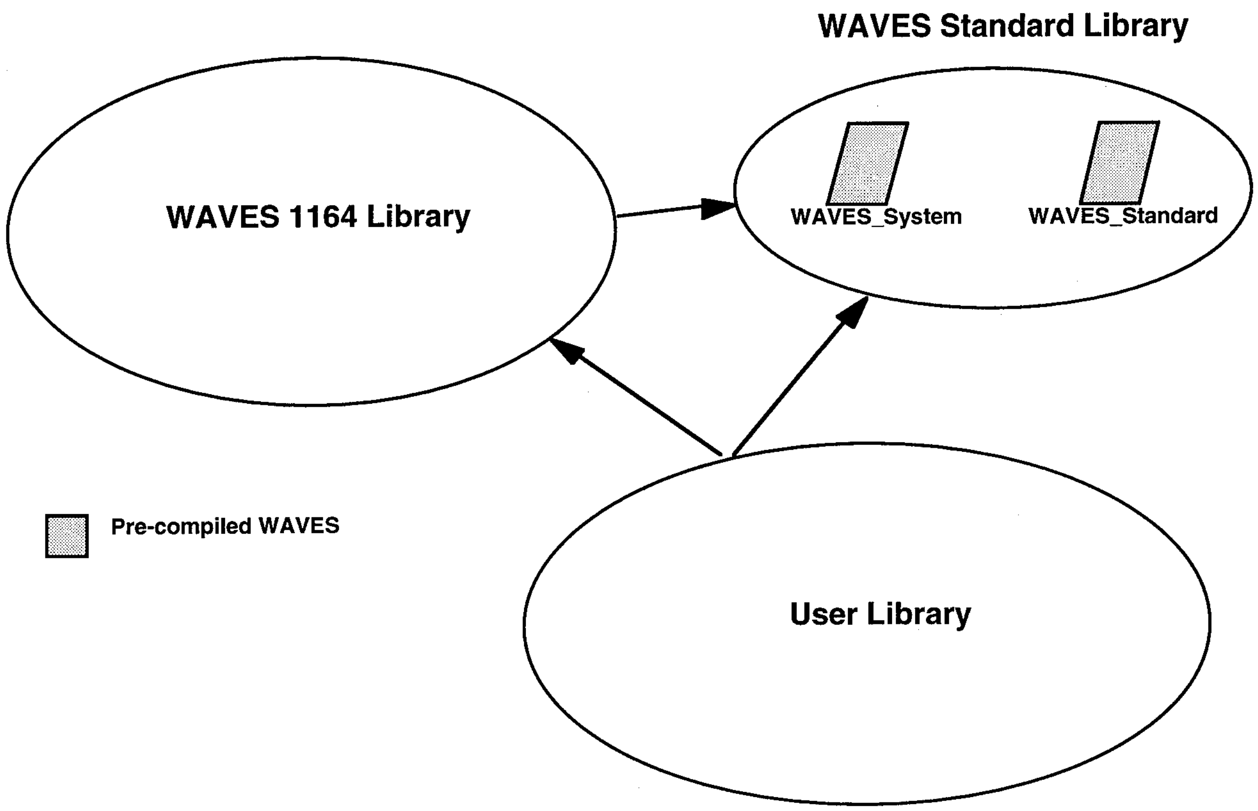


Figure 1.3.2 New WAVES library structure.

1.4 WAVES Common Packages

There are three different sets of library functions necessary to support the WAVES-VHDL testbench configuration. The library functions address IEEE Std 1164-1993 logic values, waveform shapes, and testbench utilities. Chapters 2 through 4 will describe the library functions generated to create the WAVES 1164 library. Chapters 5 and 6 will discuss the WAVES and

VHDL elements related to the user's library space.

1.4.1 WAVES 1164 Logic Values

First, a WAVES logic value system was created. These WAVES packages define a logic value system based on the IEEE Std 1164-1993 Multivalued Logic System [3]. They provide a library of reusable elements for WAVES users who are developing VHDL models that are compliant with the IEEE standard logic package. WAVES logic values define the events that occur on the waveform signals generated by the WAVES data set. This logic value system is almost identical to the Std 1164 logic values, however, they also account for the waveform signal direction which is required to generate a self monitoring testbench.

1.4.2 Waveform Shapes

A second set of library functions was created to establish a set of basic waveform shapes (formats) that may be used to construct complete, complex waveform descriptions for all of the input and output signals for a given model or unit under test (UUT). A WAVES waveform is comprised of three basic elements: pattern data (the truth table data), format (the waveform shape), and timing (the edge transition points). These three elements combine to create the waveforms for data input to the model (the drive data) and the data expected on the output of the model (the expect or compare data). A library of functions that generate common shapes was constructed to simplify the waveform generation process and provide a link to ATS utilization. The library functions are used to specify the format and timing values associated with the supplied pattern data.

1.4.3 TestBench Utilities

The third set of library functions developed provide for the integration and tool support for using WAVES in the VHDL simulation environment. This library of functions provides the means to connect the WAVES port list signals to the model and evaluate the model response for compatibility to the WAVES expected response. The library, when utilized in conjunction with the testbench tool, eliminates hand development of the testbench. The functions generated have been developed to support any testbench with uni-directional or bi-directional pins.

1.5 OVERVIEW

This report does not describe the WAVES standard, but rather builds upon a common set of utilities from which a user may build a waveform description for a specific UUT. Chapters 2 through 4 introduce and explain the basic construction of the common library packages and establishes the framework for the examples that follow. Chapter 5 discusses in detail the automation tools and how they aid a user in creating their WAVES data sets and generating a VHDL testbench for applying the stimulus and response to the models within a VHDL simulation environment. These tools were developed specifically around the library packages described in Chapters 2 through 4. Chapter 6 contains three complete model examples which demonstrate the utility of the library packages and the support tool. Each example in Chapter 6 introduces a new aspect of either the tool-generated output or the concepts supported by the library packages. Fragments of the examples appear in the text of each section with the complete VHDL model, WAVES data set and VHDL testbench appearing in the respective Appendix.

CHAPTER 2: WAVES LOGIC VALUE SYSTEM FOR IEEE STD 1164-1993

2.1 INTRODUCTION

This chapter presents and describes the WAVES packages that define a logic value system based on the IEEE Std 1164-1993, Multivalued Logic System. These packages provide a library of reusable elements for WAVES users in developing VHDL models that are compliant with the IEEE standard logic package. WAVES logic values define the events that occur on the signals of the waveform. This WAVES logic value system provides the basic building blocks for constructing the frame shapes that are described in the next chapter. This logic value system consists of three WAVES declarations that reside in two packages: WAVES_1164_Pin_Codes and WAVES_1164_Logic_Value.

2.2 WAVES_1164_PIN_CODES

The first package, WAVES_1164_Pin_Codes, contains the constant declaration of a string that enumerates all of the legal 1164 pin codes that may appear in the WAVES external (truth table) vector file. The name of this constant is **pin_codes**. The code for this package is given in listing 2.2.1.

```
PACKAGE waves_1164_pin_codes IS
    CONSTANT pin_codes : STRING := "X01ZWLH-";
END waves_1164_pin_codes;
```

Listing 2.2.1 WAVES_1164_Pin_Codes Package.

The pin codes string declaration includes all of the legal IEEE standard 1164 codes except 'U' (uninitialized). The order of the elements of the pin codes string is not significant. They have been declared in this order for consistency with IEEE Std 1164-1993.

Omitting the 'U' from the list of 1164 codes that are allowed eliminates its use in a WAVES external file. The reason is that it makes little sense to drive a model input with an uninitialized value. The uninitialized state, 'U', of a given signal is dictated by the initial state of the model during elaboration, not by the values placed on the inputs of a model by the testbench. Likewise, it makes no sense to require and verify that a given model output generates an uninitialized value. In fact, we expect the signals of the model to be uninitialized when simulation begins, but we don't actually verify that their value is 'U'. Typically, we don't look at the values of model outputs until we care what their values actually are. Therefore, there is no reason for the WAVES data set to expect a 'U' on a given signal.

2.3 WAVES_1164_Logic_Value

The second package, WAVES_1164_Logic_Value, contains an enumerated type declaration and a function declaration. The name of the enumerated type is **logic_value** and the name of the function is **value_dictionary**. The code for the package specification is given in listing 2.3.1.

```
LIBRARY waves_std;
USE waves_std.waves_standard.ALL;
PACKAGE waves_1164_logic_value IS

    TYPE logic_value IS
                (
                  dont_care,
                  sense_x,
                  sense_0,
                  sense_1,
                  sense_z,
                  sense_w,
                  sense_l,
                  sense_h,
                  drive_x,
                  drive_0,
                  drive_1,
                  drive_z,
                  drive_w,
                  drive_l,
                  drive_h );

FUNCTION value_dictionary( value : logic_value )
       RETURN event_value;

END waves_1164_logic_value;
```

Listing 2.3.1 WAVES_1164_Logic_Value Package Specification.

There are several interesting aspects to this package. First, the package is placed in the context of the WAVES_Standard package. This is required to make the **event_value** type visible. Next the **logic_value** enumerated type is declared. This enumeration type declares the names of all of the legal logic values that can be used to generate events on a waveform. The first logic value that is listed is **dont_care** which corresponds to the '-' code in Standard Logic. Next, there is a set of **sense** logic values and a set of **drive** logic values for each of the following Standard Logic codes, 'X', '0', '1', 'Z', 'W', 'L', and 'H'. Also, notice that as discussed for the WAVES_1164_Pin_Codes package, no logic value corresponds to the Standard Logic code 'U'. Finally, the **value_dictionary** function is declared. This function will be discussed below. First we will discuss the grouping of the elements of the **logic_value** declaration.

The logic values enumerated above represent three separate groupings: the **dont_care** group, the **sense** group, and the **drive** group. This grouping reflects the different nature and use of these logic values. The **dont_care** logic value appears first in the declaration for initialization reasons. After elaboration, all signals not explicitly assigned a default value have the value T'left, where T is the type mark of the signal. This causes the post elaboration values of each signal of the WAVES data set to be **dont_care**, since Logic_value'left is **dont_care**. The post elaboration values of all implicitly defaulted outputs on any model that is compliant with IEEE Std 1164-1993 will be 'U', since STD_logic'left is 'U'. When this is the case, the WAVES data set effectively states that it does not care that the model may have uninitialized outputs at the beginning of the simulation. This is consistent with the way that the model will be tested.

The **dont_care** logic value is grouped separately from the **drive** logic values, since it makes little sense to drive the model with a "don't care." Typically, when we truly don't care, the model is driven with either a '1' or a '0', not a '-'. In fact, when a resolved signal is driven with '-', the standard logic (1164) resolution function generates an 'X'. Therefore, the function of the **dont_care** logic value is for stating that we do not care what the actual output of the model is at a given time. This function differs from the **sense** logic values in that each **sense** logic value indicates the value that we expect on a given model output.

The **drive** logic values occur on the waveform whenever a signal generated by the WAVES data set represents stimulus to be "driven" on an input of the model. The **sense** logic values occur on the waveform whenever a signal generated by the WAVES data set represents the expected response of the model. This "dual" logic value system allows WAVES to represent the direction of the signals that make up the waveform and is necessary information for verification.

The code for the WAVES_1164_Logic_Value package is given in Appendix A1.2. The package body consists only of the implementation of the **value_dictionary** function. This function is used to document the meaning of each element enumerated in the logic value type declaration with regard to state, strength, direction, and relevance. This function is not used by VHDL when simulating a WAVES data set. The only purpose of the **value_dictionary** function is to document the semantics of the logic values. The definition of the state and strength values associated with each logic value are derived from the IEEE Std 1164-1993 definitions.

CHAPTER 3: WAVES FRAME SET FORMATS

3.1 INTRODUCTION

In order to verify the functionality and performance of a digital design, an analysis system must supply a set of input signals and know the expected outputs which will come from the unit and know when the outputs are expected to come (in order to flag any errors). This analysis system may simply be a VHDL simulator used for design verification or a complex automated test system used for electrical characterization. This chapter describes the WAVES packages that define a set of basic waveform formats (shapes) that may be used to construct complete, complex waveform descriptions for all the input and output signals for a given unit under test (UUT). These packages provide a library of reusable waveform shapes for WAVES users when developing VHDL models that are compliant with the WAVES_1164_Logic_Value package described in Chapter 2. This WAVES package provides the basic building blocks for constructing UUT waveforms. The waveform package WAVES_1164_FRAMES is included in Appendix 2.

3.2 TERMINOLOGY

A waveform is comprised of three basic elements: **pattern data** (the truth table data), **format** (the waveform shape), and **timing** (the edge transition points). These three elements combine to create the waveforms for data input to the UUT (the **drive** data) and the data expected on the output of the UUT (the **expect** or **compare** data).

The pattern data refers to the legal pin codes that are valid in a WAVES external file as defined in Chapter 2. The pattern data 'X', '0', '1', 'Z', 'W', 'L', 'H', or '-' occurs once for each WAVES **slice** (which roughly corresponds to a tester cycle). By itself, the pattern data means nothing to the UUT, because the data must be formatted. It is the job of the WAVES frame set descriptions to supply the information necessary for the analysis system to properly "build" data sheet-like waveforms. The pattern data is applied to the UUT in the chosen format(s) at the times specified in order to create waveform(s) at the UUT pins.

The creation of a WAVES formatted signal is comprised of several WAVES elements that are built up to define a static definition of the waveform shape and associated timing. This chapter describes the WAVES_1164_FRAMES package developed to establish a basic set of waveform shapes that are readily available to the user. In order to describe these waveforms, we must first define some basic terminology.

The time segments associated with a waveform can be thought of as consecutive advancing slices of a waveform. These segments of time are called WAVES **slices** and roughly correspond to test system "cycles". A tester **cycle** is one tester period and is measured from t0 (time zero) to t0 of the next cycle (see Figure 3.2.1).

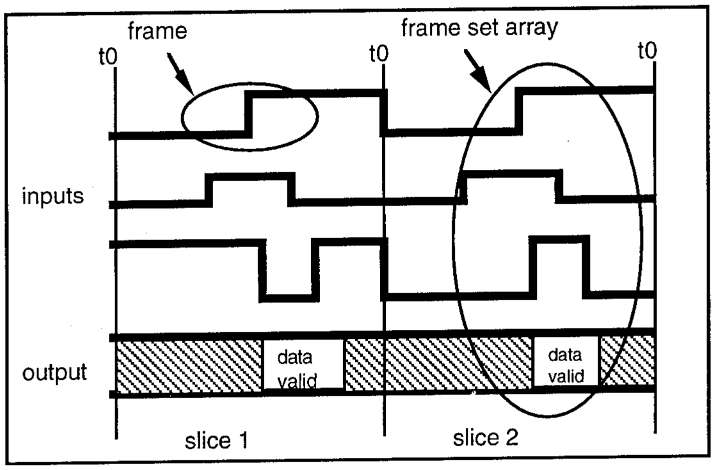


Figure 3.2.1 Sample Input/Output Waveforms and WAVES Slices.

Typically, each individual waveform may have multiple edge transitions in one period or cycle. This list of edge transitions (WAVES events) on an individual signal of the waveform within a slice is defined as a **frame**. The set of frames for all possible legal pattern values (WAVES pin codes) that can be used on a signal is called a **frame set**. The functions that are described in this chapter are used to define and establish a common set of the waveform shapes (WAVES frame sets). In Chapter 6 we will show how these waveform shapes are combined to create a **frame set array** to describe the waveform signals on all the UUT inputs and outputs during a slice. For the rest of this chapter we will refer to the pre-defined collection of waveform shapes or frame sets as formats.

Drive formats are designated as the waveform shapes that are associated with a UUT input signal. That is, what the signal does during the slice when the pin code is specified. This WAVES_1164_FRAMES package contains a reasonable, generic set of drive formats for the 1164 logic values that may be used by anyone. The WAVES drive frame formats we have developed are: **non-return** (NR), **return high** (RH), **return low** (RL), **pulse high** (PH), **pulse high skew** (PHS), **pulse low** (PL), **pulse low skew** (PLS), and **surround by complement** (SC). The WAVES_1164_FRAMES package contains a function which corresponds to each format named (see section 3.4).

For expected data (output signals), two WAVES format functions were developed to supply the analysis system with the information necessary to compare the *actual* unit under test output with the *expected* output at a specified time. The compare is performed over a span of time by using a **window** compare or a **window skew** compare function (see section 3.5).

The WAVES frames developed and described in this chapter provide great flexibility and may be used to build very complex waveforms. The package specification that defines these frame sets is shown in Listing 3.2.1.

```
LIBRARY waves_1164;
USE waves_1164.waves_1164_pin_codes.ALL;
USE waves_1164.waves_1164_logic_value.ALL;
USE waves_1164.waves_interface.ALL;
PACKAGE waves_1164_frames IS

  --
  -- Declare functions that return Frame Sets.
  --

   FUNCTION non_return( t1 : TIME ) RETURN frame_set;
   FUNCTION return_low( t1, t2 : TIME ) RETURN frame_set;
   FUNCTION return_high( t1, t2 : TIME ) RETURN frame_set;
   FUNCTION surround_complement( t1, t2 : TIME ) RETURN frame_set;
   FUNCTION pulse_low( t1, t2 : TIME ) RETURN frame_set;
   FUNCTION pulse_low_skew( t0, t1, t2 : TIME ) RETURN frame_set;
   FUNCTION pulse_high( t1, t2 : TIME ) RETURN frame_set;
   FUNCTION pulse_high_skew( t0, t1, t2 : TIME ) RETURN frame_set;
   FUNCTION window( t1, t2 : TIME ) RETURN frame_set;
   FUNCTION window_skew( t0, t1, t2 : TIME ) RETURN frame_set;

END waves_1164_frames;
```

Listing 3.2.1 WAVES_1164_frames package declaration.

3.3 Pattern Data Dependency

As mentioned before, usually, for each individual waveform signal there is just one pair of timing edge transitions: the **leading** edge transition t1 and the **trailing** edge transition t2. The formats developed in this package conform to this convention. The waveform shapes are entirely dependent upon the data values present in the pattern vectors. Data dependent formats always present pattern data at least between the t1 and t2 markers. The format selected also describes the logic level for the time from t2 until t1 of the next cycle (and in some cases the format selected dictates the logic level from t0 at the beginning of the cycle until the leading edge transition time t1).

3.4 Drive Formats

3.4.1 Compound Formats

Non-Return (NR). Typical output from a simulation program is in NR format, which is the simplest way to represent UUT behavior. The NR frame forces a pattern data transition at t1 (only), and continues driving data until the next t1, ignoring the subsequent t2 and t0. That is, at t0, the drive level is whatever the data of the previous cycle had been. At t1, it drives the data level of the present cycle and remains at that level at least until t1 of the following cycle, ignoring both t2 and the subsequent t0. A sample waveform showing the NR format is shown in Figure 3.4.1. The NR frame set definition is shown in Listing 3.4.1.

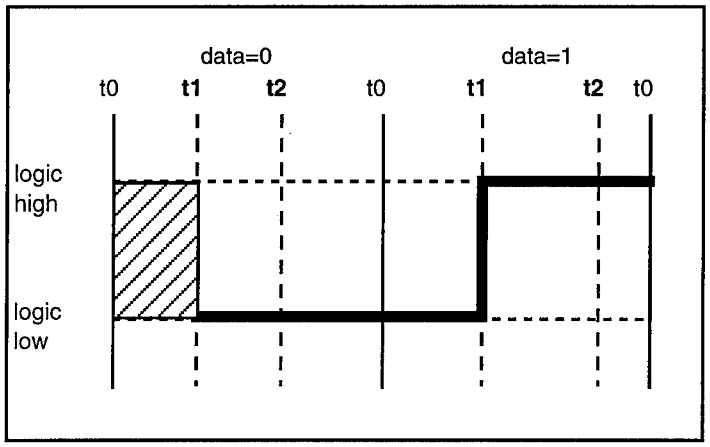


Figure 3.4.1 Non-Return Waveform.

```
FUNCTION non_return( t1 : TIME ) RETURN frame_set IS

   CONSTANT edge : event_time := etime( t1 );

BEGIN
  RETURN
     new_frame_set( 'X', frame_event( (drive_X, edge) ) ) +
     new_frame_set( '0', frame_event( (drive_0, edge) ) ) +
     new_frame_set( '1', frame_event( (drive_1, edge) ) ) +
     new_frame_set( 'Z', frame_event( (drive_Z, edge) ) ) +
     new_frame_set( 'W', frame_event( (drive_W, edge) ) ) +
     new_frame_set( 'L', frame_event( (drive_L, edge) ) ) +
     new_frame_set( 'H', frame_event( (drive_H, edge) ) ) +
     new_frame_set( '-', frame_event );
END non_return;
```

Listing 3.4.1 Non-Return Frame set declaration.

Return High (RH). The RH frame drives the specified data level from t1 to t2 and drives the level high from t2 to the following t0. That is, the drive level is at the value that was present at the end of the previous cycle from t0 to t1, transitions at t1 to the valid data pattern level, and goes high (or stays high depending on the pattern data value) at time t2. A sample waveform showing the RH format is shown in Figure 3.4.2. The heavy dashed line represents the drive level present due to the frame set definition and not from the pattern data shown at the top of the figure. The RH frame set definition is shown in Listing 3.4.2.

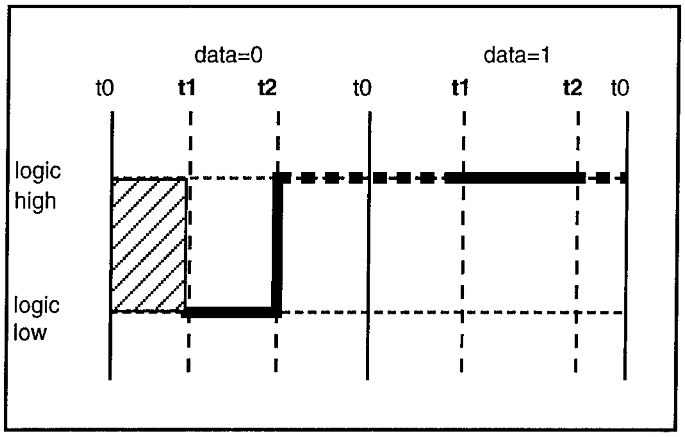


Figure 3.4.2 Return High Waveform.

```
FUNCTION return_high( t1, t2 : TIME ) RETURN frame_set IS

  CONSTANT edge1 : event_time := etime( t1 );
  CONSTANT edge2 : event_time := etime( t2 );

BEGIN
  ASSERT t1 < t2
  REPORT "Timing violation in Return_High frames." &
         "The inequality: T1 < T2 Must hold."
  SEVERITY FAILURE;
  RETURN
     new_frame_set( 'X', frame_elist( ((drive_X, edge1),
                                       (drive_1, edge2)) ) ) +
     new_frame_set( '0', frame_elist( ((drive_0, edge1),
                                       (drive_1, edge2)) ) ) +
     new_frame_set( '1', frame_event( ( drive_1, edge1)   ) ) +
     new_frame_set( 'Z', frame_elist( ((drive_Z, edge1),
                                       (drive_1, edge2)) ) ) +
     new_frame_set( 'W', frame_elist( ((drive_W, edge1),
                                       (drive_1, edge2)) ) ) +
     new_frame_set( 'L', frame_elist( ((drive_L, edge1),
                                       (drive_1, edge2)) ) ) +
     new_frame_set( 'H', frame_elist( ((drive_H, edge1),
                                       (drive_1, edge2)) ) ) +
     new_frame_set( '-', frame_event( ( drive_1, edge2 ) ) );
END return_high;
```

Listing 3.4.2 Return High Frame set declaration.

Return Low (RL). The RL frame drives the specified data level from t1 to t2 and drives the level low from t2 to the following t0. That is, the drive level is at the value that was present at the end of the previous cycle from t0 to t1, transitions at t1 to the valid data pattern level, and goes low (or stays low depending on the pattern data value) at time t2. A sample waveform showing the RL format is shown in Figure 3.4.2. The heavy dashed line represents the drive level present due to the frame set definition and not from the pattern data shown at the top of the figure. The RL frame set definition is shown in Listing 3.4.3.

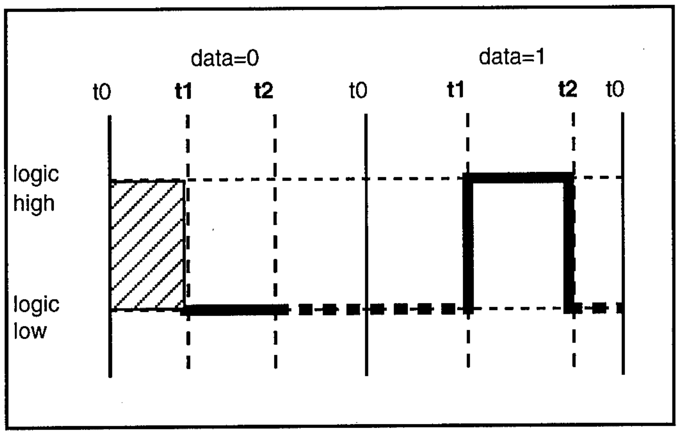


Figure 3.4.3 Return Low Waveform.

```
FUNCTION return_low( t1, t2 : TIME ) RETURN frame_set IS

  CONSTANT edge1 : event_time := etime( t1 );
  CONSTANT edge2 : event_time := etime( t2 );

BEGIN
  ASSERT t1 < t2
  REPORT "Timing violation in Return_Low frames." &
         "The inequality : T1 < T2 Must hold."
  SEVERITY FAILURE;
  RETURN
     new_frame_set( 'X', frame_elist( ((drive_X, edge1),
                                        (drive_0, edge2)) ) ) +
     new_frame_set( '0', frame_event( ( drive_0, edge1)   ) ) +
     new_frame_set( '1', frame_elist( ((drive_1, edge1),
                                        (drive_0, edge2)) ) ) +
     new_frame_set( 'Z', frame_elist( ((drive_Z, edge1),
                                        (drive_0, edge2)) ) ) +
     new_frame_set( 'W', frame_elist( ((drive_W, edge1),
                                        (drive_0, edge2)) ) ) +
     new_frame_set( 'L', frame_elist( ((drive_L, edge1),
                                        (drive_0, edge2)) ) ) +
     new_frame_set( 'H', frame_elist( ((drive_H, edge1),
                                        (drive_0, edge2)) ) ) +
     new_frame_set( '-', frame_event( ( drive_0, edge2 ) ) );
END return_low;
```

Listing 3.4.3 Return Low Frame set declaration.

Surround by Complement (SC). The SC frame drives the complement of the pattern data from t0 to t1 and from t2 to the following t0. That is, the drive level is at the complement of the pattern data at the beginning of the cycle, transitions to the pattern data level at t1, and returns to the complement at t2. Many test and analysis systems refer to this format as “Return to Complement” even though surround-by-complement is a more accurate description. Note that when t2 is in a subsequent cycle (slice), data returns to the complement of the pattern data of the new cycle (not the complement of the data of the original cycle).). Also note that the SC format is only applicable to data values of “1”, “0”, “H”, and “L”, for all other data values the drive levels are unchanged from the previous cycle. A sample waveform showing the SC format is shown in Figure 3.4.4. The heavy dashed line represents the drive level present due to the frame set definition and not from the pattern data. The SC frame set definition is shown in Listing 3.4.4.

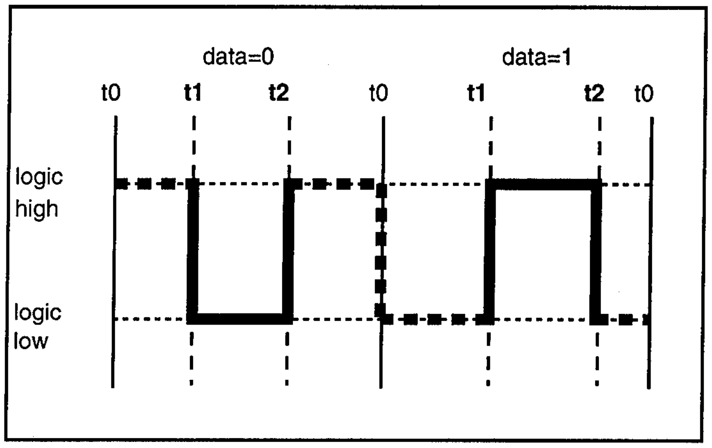


Figure 3.4.4 Surround by Complement Waveform.

```
FUNCTION surround_complement( t1, t2 : TIME) RETURN frame_set IS

   CONSTANT edge0 : event_time := etime( 0 ns );
   CONSTANT edge1 : event_time := etime( t1 );
   CONSTANT edge2 : event_time := etime( t2 );

BEGIN
   ASSERT t1 < t2
   REPORT "Timing violation in Surround_Complement frames.  " &
          "The inequality: T1 < T2 Must hold."
   SEVERITY FAILURE;
   RETURN
      new_frame_set( 'X', frame_event( ( drive_X, edge1)   ) ) +
      new_frame_set( '0', frame_elist( ((drive_1, edge0),
                                         (drive_0, edge1),
                                         (drive_1, edge2)) ) ) +
      new_frame_set( '1', frame_elist( ((drive_0, edge0),
                                         (drive_1, edge1),
                                         (drive_0, edge2)) ) ) +
      new_frame_set( 'Z', frame_event( ( drive_Z, edge1)   ) ) +
      new_frame_set( 'W', frame_event( ( drive_W, edge1)   ) ) +
      new_frame_set( 'L', frame_elist( ((drive_H, edge0),
                                         (drive_l, edge1),
                                         (drive_H, edge2)) ) ) +
      new_frame_set( 'H', frame_elist( ((drive_L, edge0),
                                         (drive_h, edge1),
                                         (drive_L, edge2)) ) ) +
      new_frame_set( '-', frame_event );
END surround_complement;
```

Listing 3.4.4 Surround by Complement Frame set declaration.

3.4.2 Pulse Formats

Pulse formats drive a fixed waveform to the unit under test which are useful for supplying clock signals to the UUT. There are two pulse formats; Pulse Low and Pulse High. In addition, these frame set formats have two separate instantiations. The first instantiation (Pulse_Low and Pulse_High) allow for data pulses only within one WAVES slice (t0 is always set to 0 ns and t1 and t2 occur in the same slice). The second instantiation (Pulse_Low_Skew and Pulse_High_Skew) allows the data pulse to be present across WAVES slice boundaries (t0 does not have to be at 0 ns and t2 is not in the same slice as t1 which introduces the concept of t0' as shown in Figure 3.4.6 and 3.4.8 below). The following descriptions and examples will attempt to clarify this concept.

Pulse Low/Pulse Low Skew (PL/PLS). If the vector pattern data contains either an "L" or a "0" this format will drive a high level at t0 (or t0'), drive to a low level on the first edge (t1), then drive a high level at the second edge (t2) until the end of the cycle. That is, drive a high level at t0, transition high-to-low at t1 and low-to-high at t2. Note that for the PLS format the entire waveform is shifted into the next cycle by an amount depicted by t0', so that t2 occurs in a subsequent slice (cycle). The signal will stay high for the entire cycle for vector pattern data values of '1' and 'H', and the signal remains at the previous cycle value for all other data formats. Sample waveforms showing the PL format are shown in Figure 3.4.5 and the PLS format in Figure 3.4.6. The PL frame set definition is shown in Listing 3.4.5 and the PLS frame set definition is shown in Listing 3.4.6.

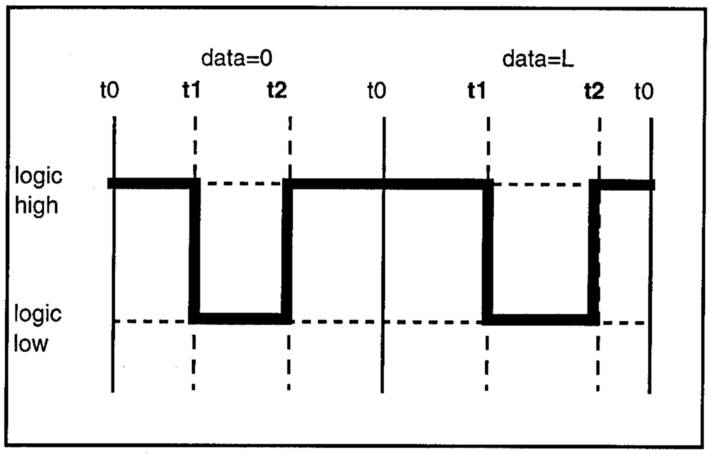


Figure 3.4.5 Pulse Low Waveform.

```
FUNCTION pulse_low( t1, t2 : TIME ) RETURN frame_set IS

   CONSTANT edge0 : event_time := etime( 0 ns );
   CONSTANT edge1 : event_time := etime( t1 );
   CONSTANT edge2 : event_time := etime( t2 );

BEGIN
   ASSERT t1 < t2
  REPORT "Timing violation in Pulse_Low frames." &
         "The inequality: T1 < T2 Must hold."
  SEVERITY FAILURE;
  RETURN
     new_frame_set( 'X', frame_event ) +
     new_frame_set( '0', frame_elist( ((drive_1, edge0),
                                        (drive_0, edge1),
                                        (drive_1, edge2)) ) ) +
     new_frame_set( '1', frame_event( ( drive_1, edge0)   ) ) +
     new_frame_set( 'Z', frame_event ) +
     new_frame_set( 'W', frame_event ) +
     new_frame_set( 'L', frame_elist( ((drive_H, edge0),
                                        (drive_L, edge1),
                                        (drive_H, edge2)) ) ) +
     new_frame_set( 'H', frame_event( ( drive_H, edge0)   ) ) +
     new_frame_set( '-', frame_event );
END pulse_low;
```

Listing 3.4.5 Pulse Low Frame set declaration.

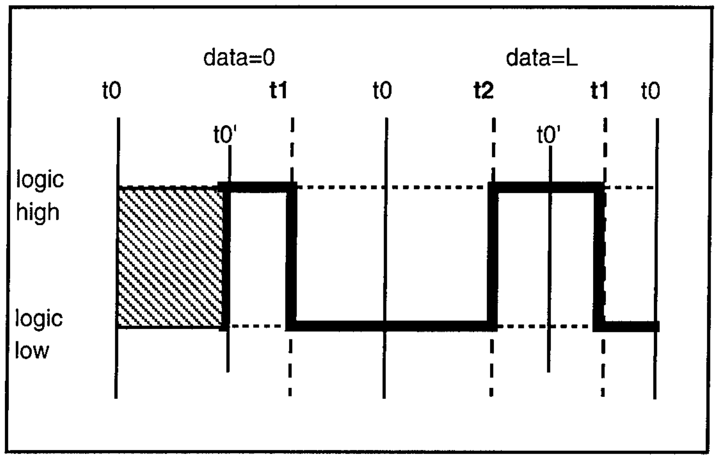


Figure 3.4.6 Pulse Low Skew Waveform.

```
FUNCTION pulse_low_skew( t0, t1, t2 : TIME ) RETURN frame_set IS

     CONSTANT edge0 : event_time := etime( t0 );
     CONSTANT edge1 : event_time := etime( t1 );
     CONSTANT edge2 : event_time := etime( t2 );

  BEGIN
     ASSERT t0 < t1 AND t1 < t2
    REPORT "Timing violation in Pulse_Low frames." &
           "The inequality: T0 < T1 < T2 Must hold."
    SEVERITY FAILURE;
    RETURN
       new_frame_set( 'X', frame_event ) +
       new_frame_set( '0', frame_elist( ((drive_1, edge0),
                                          (drive_0, edge1),
                                          (drive_1, edge2)) ) ) +
       new_frame_set( '1', frame_event( ( drive_1, edge0)   ) ) +
       new_frame_set( 'Z', frame_event ) +
       new_frame_set( 'W', frame_event ) +
       new_frame_set( 'L', frame_elist( ((drive_H, edge0),
                                          (drive_L, edge1),
                                          (drive_H, edge2)) ) ) +
       new_frame_set( 'H', frame_event( ( drive_H, edge0)   ) ) +
       new_frame_set( '-', frame_event );
  END pulse_low_skew;
```

Listing 3.4.6 Pulse Low Skew Frame set declaration.

Pulse High/Pulse High Skew (PH/PHS). If the vector pattern data contains either an "H" or a "1" this format will drive a low level at t0 (or t0'), drive to a high level on the first edge (t1), then drive a low level at the second edge (t2). That is, drive a low level at t0, transition low-to-high at t1 and high-to-low at t2. Note that (like the PLS format described previously) for the PHS format the entire waveform is shifted into the next cycle by an amount depicted by t0', so t2 occurs in a subsequent slice (cycle). The signal will stay low for the entire cycle for vector pattern data values of “0” and “L”, and the signal remains at the previous cycle value for all other data formats. Sample waveforms showing the PH format are shown in Figure 3.4.7 and the PHS format in Figure 3.4.8. The PH frame set definition is shown in Listing 3.4.7 and the PHS frame set definition is shown in Listing 3.4.8.

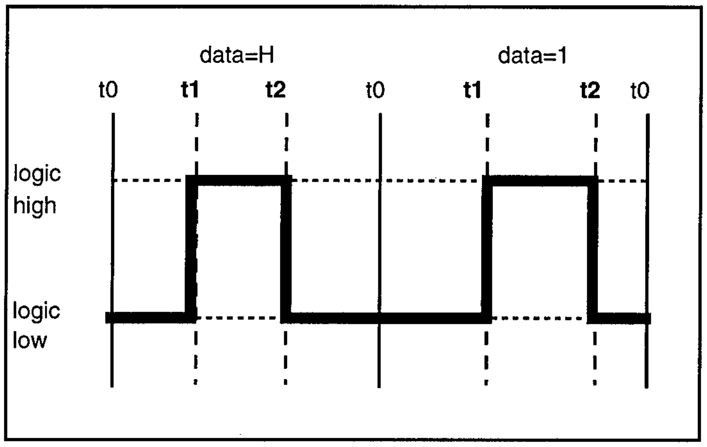


Figure 3.4.7 Pulse High Waveform.

```
FUNCTION pulse_high( t1, t2 : TIME ) RETURN frame_set IS

  CONSTANT edge0 : event_time := etime( 0 ns );
  CONSTANT edge1 : event_time := etime( t1 );
  CONSTANT edge2 : event_time := etime( t2 );

BEGIN
  ASSERT t1 < t2
  REPORT "Timing violation in Pulse_High frames." &
         "The inequality: T1 < T2 Must hold."
  SEVERITY FAILURE;
  RETURN
    new_frame_set( 'X', frame_event ) +
    new_frame_set( '0', frame_event( ( drive_0, edge0)   ) ) +
    new_frame_set( '1', frame_elist( ((drive_0, edge0),
                                      (drive_1, edge1),
                                      (drive_0, edge2)) ) ) +
    new_frame_set( 'Z', frame_event ) +
    new_frame_set( 'W', frame_event ) +
    new_frame_set( 'L', frame_event( ( drive_L, edge0)   ) ) +
    new_frame_set( 'H', frame_elist( ((drive_L, edge0),
                                      (drive_H, edge1),
                                      (drive_L, edge2)) ) ) +
    new_frame_set( '-', frame_event );
END pulse_high;
```

Listing 3.4.7 Pulse High Frame set declaration (first instantiation).

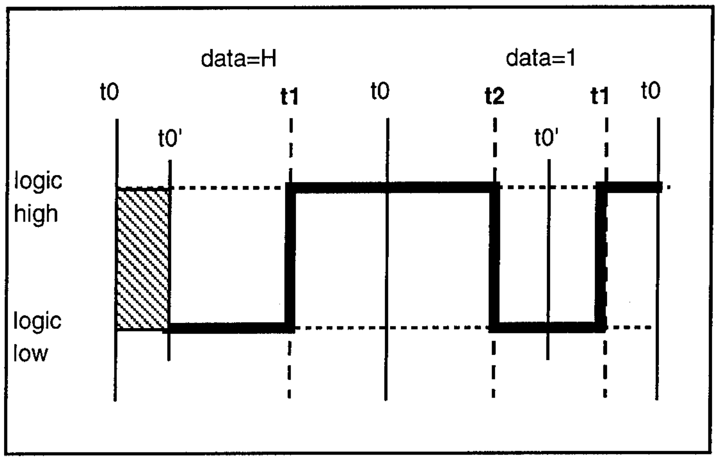


Figure 3.4.8 Pulse High Skew Waveform.

```
FUNCTION pulse_high_skew( t0, t1, t2 : TIME ) RETURN frame_set IS

   CONSTANT edge0 : event_time := etime( t0 );
   CONSTANT edge1 : event_time := etime( t1 );
   CONSTANT edge2 : event_time := etime( t2 );

BEGIN
   ASSERT t0 < t1 AND t1 < t2
  REPORT "Timing violation in Pulse_High frames." &
         "The inequality: T0 < T1 < T2 Must hold."
  SEVERITY FAILURE;
  RETURN
     new_frame_set( 'X', frame_event ) +
     new_frame_set( '0', frame_event( ( drive_0, edge0)  ) ) +
     new_frame_set( '1', frame_elist( ((drive_0, edge0),
                                       (drive_1, edge1),
                                       (drive_0, edge2)) ) ) +
     new_frame_set( 'Z', frame_event ) +
     new_frame_set( 'W', frame_event ) +
     new_frame_set( 'L', frame_event( ( drive_L, edge0)  ) ) +
     new_frame_set( 'H', frame_elist( ((drive_L, edge0),
                                       (drive_H, edge1),
                                       (drive_L, edge2)) ) ) +
     new_frame_set( '-', frame_event );
END pulse_high_skew;
```

Listing 3.4.8 Pulse High Frame set declaration (second instantiation).

3.5 Expected Output Formats

The **Window** format starts the valid output data window at the t1 edge and ends the valid data window at the subsequent t2 edge. That is, the output data is valid for the entire duration from t1 to t2. The output data is set to "don't care" from t0 to t1 and from t2 to the subsequent t0. An example waveform showing Window formats where the data expected on the output of the UUT is low, high, and then midband is shown in Figure 3.5.1. The Window frame set definition is shown in Listing 3.5.1.

The **Window Skew** format is similar to the Window format except that the t2 edge occurs in a subsequent cycle (slice), delayed by the time t0', much like the PHS and PLS formats discussed in section 3.4.2. An example waveform showing Window Skew formats where the data expected on the output of the UUT is low, high, and then midband is shown in Figure 3.5.2. The Window Skew frame set definition is shown in Listing 3.5.2.

As stated previously, the window for valid data opens at time t1 and closes at time t2. Many analysis systems also have the capability of sampling UUT output data at a single point in time with what is referred to as an **edge strobe.** Using the window formats described above, it is possible to mimic an edge strobe by setting the t1 and t2 times to be as close to one another as possible, constrained only by the timing resolution of the analysis system. It should be noted that this is not the same as using the test system's hardware edge strobe capability, and the results will not be as accurate.

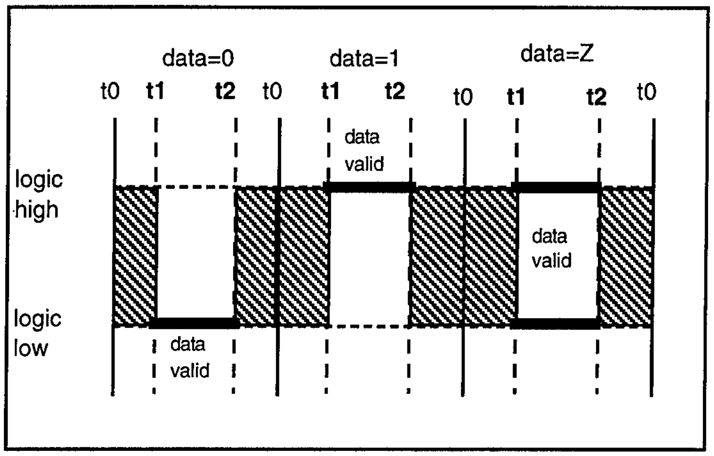


Figure 3.5.1 Window Strobe Waveform.

```
 FUNCTION window( t1, t2 : TIME ) RETURN frame_set IS

    CONSTANT edge0 : event_time := etime( 0 ns );
    CONSTANT edge1 : event_time := etime( t1 );
    CONSTANT edge2 : event_time := etime( t2 );

BEGIN
   ASSERT t1 < t2
   REPORT "Timing violation in Window frames." &
          "The inequality: T1 < T2 Must hold."
   SEVERITY FAILURE;
   RETURN
      new_frame_set( 'X', frame_elist( ((dont_care, edge0),
                                         (sense_X, edge1),
                                         (dont_care, edge2)) ) ) +
      new_frame_set( '0', frame_elist( ((dont_care, edge0),
                                         (sense_0, edge1),
                                         (dont_care, edge2)) ) ) +
      new_frame_set( '1', frame_elist( ((dont_care, edge0),
                                         (sense_1, edge1),
                                         (dont_care, edge2)) ) ) +
      new_frame_set( 'Z', frame_elist( ((dont_care, edge0),
                                         (sense_Z, edge1),
                                         (dont_care, edge2)) ) ) +
      new_frame_set( 'W', frame_elist( ((dont_care, edge0),
                                         (sense_W, edge1),
                                         (dont_care, edge2)) ) ) +
      new_frame_set( 'L', frame_elist( ((dont_care, edge0),
                                         (sense_L, edge1),
                                         (dont_care, edge2)) ) ) +
      new_frame_set( 'H', frame_elist( ((dont_care, edge0),
                                         (sense_H, edge1),
                                         (dont_care, edge2)) ) ) +
      new_frame_set( '-', frame_event( ( dont_care, edge0)  ) );
END window;
```

Listing 3.5.1 Window Frame set declaration.

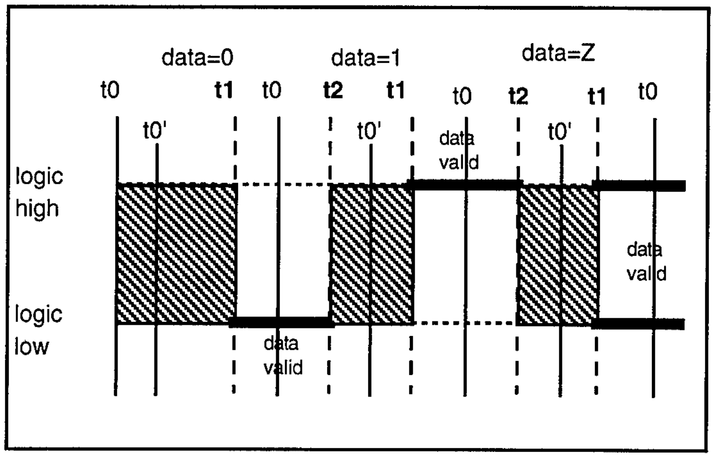


Figure 3.5.2 Window Skew Strobe Waveforms.

```
FUNCTION window_skew( t0, t1, t2 : TIME ) RETURN frame_set IS

   CONSTANT edge0 : event_time := etime( t0 );
   CONSTANT edge1 : event_time := etime( t1 );
   CONSTANT edge2 : event_time := etime( t2 );

BEGIN
   ASSERT t0 < t1 AND t1 < t2
  REPORT "Timing violation in Window frames." &
         "The inequality: T0 < T1 < T2 Must hold."
  SEVERITY FAILURE;
  RETURN
     new_frame_set( 'X', frame_elist( ((dont_care, edge0),
                                        (sense_X, edge1),
                                        (dont_care, edge2)) ) ) +
     new_frame_set( '0', frame_elist( ((dont_care, edge0),
                                        (sense_0, edge1),
                                        (dont_care, edge2)) ) ) +
     new_frame_set( '1', frame_elist( ((dont_care, edge0),
                                        (sense_1, edge1),
                                        (dont_care, edge2)) ) ) +
     new_frame_set( 'Z', frame_elist( ((dont_care, edge0),
                                        (sense_Z, edge1),
                                        (dont_care, edge2)) ) ) +
     new_frame_set( 'W', frame_elist( ((dont_care, edge0),
                                        (sense_W, edge1),
                                        (dont_care, edge2)) ) ) +
     new_frame_set( 'L', frame_elist( ((dont_care, edge0),
                                        (sense_L, edge1),
                                        (dont_care, edge2)) ) ) +
     new_frame_set( 'H', frame_elist( ((dont_care, edge0),
                                        (sense_H, edge1),
                                        (dont_care, edge2)) ) ) +
     new_frame_set( '-', frame_event( ( dont_care, edge0)   ) );
END window_skew;
```

Listing 3.5.2 Window Skew Frame set declaration.

CHAPTER 4: WAVES TESTBENCH UTILITIES PACKAGE FOR IEEE STD 1164

4.1 INTRODUCTION

This chapter describes the VHDL package developed to support VHDL model simulation and verification utilizing a WAVES data set. This package builds upon the WAVES STD 1164 multi-logic package defined in Chapter 2 and provides the user with a seamless interface when utilizing VHDL and WAVES together. This set of library functions was developed to provide for the integration and tool support for using WAVES in a VHDL simulation environment.

These library functions provide for two different testbench aspects: first, to connect the WAVES port list signals to the model and second, to evaluate the model response for compatibility to the WAVES expected response. The premise of the functions provided in this library is that, when utilized in conjunction with the testbench generation tool, all the detailed work in developing the testbench is eliminated. The functions developed support a simple testbench with uni-directional pins, as well as a complex bi-directional design entity.

A prototype automatic testbench generation tool has also been developed based on these library elements. The purpose of the tool is to provide a methodology which minimizes the work involved by an individual in developing testbenches through the use of a common user library and a building block approach. The user is not restricted by this methodology, which is meant to eliminate repetitive work and reduce the learning cycle for WAVES. The tool provides the user with a testbench that is automatically generated. The testbench not only applies the WAVES stimulus to the model, but verifies the validity of the expected responses.

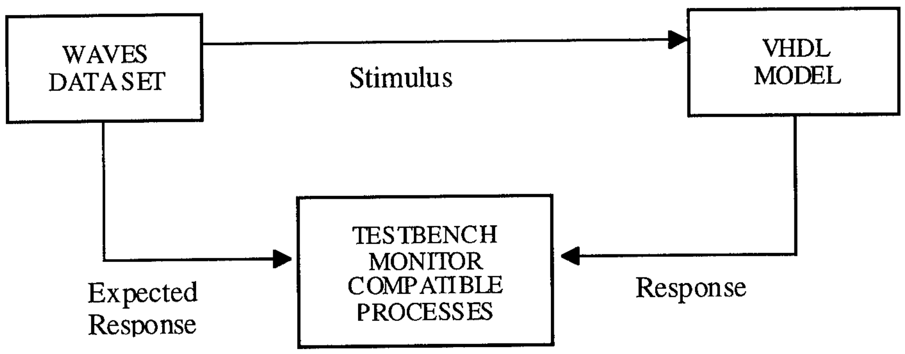


Figure 4.1.1 Functionality.

As mentioned previously, the purpose of the testbench utilities is to aid a user in developing a testbench for applying the WAVES stimulus and response to the models. Figure 4.1.1 illustrates the conceptual view of the functionality of the testbench methodology using these packages. The stimulus waveforms which are produced by the WAVES data set are applied to the VHDL model. The model then produces its' output response based on the stimulus presented. The testbench contains a set of monitoring processes which verify the expected and actual response values by ensuring that they conform to the timing as specified in the WAVES data set.

The WAVES_1164_UTILITIES contains the **stim_1164**, **expect_1164**, and **bi_dir_1164** functions to provide translation between the internal WAVES logic values, as described in Chapter 2, and the IEEE Std 1164-1993 values in the users' model. The package also contains an overloaded function **compatible** which allows a user to check or verify if the predicted or expected response is "compatible" with the actual results generated by the model (see section 4.3 for further discussion of the compatible function). In order to use this package, all of the VHDL design units must be compliant with the IEEE standard logic package. Sections 4.2 and 4.3 describe all of the functions provided in the testbench utility package. Chapter 6 contains three complete testbench usage examples, each illustrating a different aspect of the packages. The package declarations that define the functions is shown in Listing 4.1.1. The complete implementation of the WAVES_1164_UTILITIES package is included in Appendix 3.

```
PACKAGE waves_1164_utilities IS
  -----------------------------------------
  -- Procedure and Function Declarations --
  -----------------------------------------
  --
FUNCTION stim_1164( port_element : system_waves_port)
        RETURN STD_LOGIC;
FUNCTION stim_1164( port_list : system_waves_port_list)
        RETURN STD_ULOGIC_VECTOR;
FUNCTION stim_1164( port_list : system_waves_port_list)
FUNCTION expect_1164( port_element : system_waves_port)
        RETURN STD_ULOGIC;
FUNCTION expect_1164( port_list : system_waves_port_list)
        RETURN STD_ULOGIC_VECTOR;
FUNCTION bi_dir_1164( port_element : system_waves_port)
        RETURN STD_LOGIC;
FUNCTION bi_dir_1164( port_list : system_waves_port_list)
        RETURN STD_LOGIC_VECTOR;
FUNCTION compatible( actual:  STD_LOGIC;
                     expected : STD_ULOGIC )
        RETURN BOOLEAN;
FUNCTION compatible( actual:  STD_ULOGIC_VECTOR;
                     expected : STD_ULOGIC_VECTOR)
        RETURN BOOLEAN;
FUNCTION compatible( actual:  STD_LOGIC_VECTOR;
                     expected : STD_ULOGIC_VECTOR)
        RETURN BOOLEAN;
END waves_1164_utilities;
```

Listing 4.1.1 WAVES_1164_UTILITIES Declarations Section.

4.2 WAVES TRANSLATION FUNCTIONS

When used in simulation, the WAVES data set generates waveforms based on the WAVES logic values described in Chapter 2. These generated WAVES port values are not compatible with the IEEE Std 1164-1993 logic values. Instead, the WAVES ports values generated during simulation are integer values. Therefore, translations are required between the WAVES port values and the models' logic values. These WAVES integer values correspond to the position of the logic value enumeration as defined in Chapter 2. The mapping between the enumerated logic and the integer values generated on the WAVES port is shown in columns 1 and 2 of Table 4.2.1.

WAVES LOGIC VALUE	Waves Port Value	stim_1164 Value	expect_1164 Value	bi_dir_1164 Value
DONT_CARE	0	'-'	'-'	'Z'
SENSE_X	1	'X'	'X'	'Z'
SENSE_0	2	'0'	'0'	'Z'
SENSE_1	3	'1'	'1'	'Z'
SENSE_Z	4	'Z'	'Z'	'Z'
SENSE_W	5	'W'	'W'	'Z'
SENSE_L	6	'L'	'L'	'Z'
SENSE_H	7	'H'	'H'	'Z'
DRIVE_X	8	'X'	'-'	'X'
DRIVE_0	9	'0'	'-'	'0'
DRIVE_1	10	'1'	'-'	'1'
DRIVE_Z	11	'Z'	'-'	'Z'
DRIVE_W	12	'W'	'-'	'W'
DRIVE_L	13	'L'	'-'	'L'
DRIVE_H	14	'H'	'-'	'H'

Table 4.2.1 STIM_1164 Mappings.

The three overloaded functions **stim_1164, bi_dir_1164,** and **expect_1164** provide for translation of the WAVES port values into the IEEE Std 1164-1993 logic values used in a model. Table 4.2.1 shows the mapping values and the results that will be generated by the associated translation function. Figure 4.2.1 is an illustration of a testbench that would be developed for design analysis. The testbench is capable of supporting the analysis of any complex bit-level design utilizing STD logic uni-directional or bi-directional pins. The illustration contains labeled blocks that identify where the functions **stim_1164, bi_dir_1164,** and **expect_1164** would be utilized in supporting logic value translations within the testbench. The examples in Chapter 6 provide further description as to how these functions are used in the construction of the testbench.

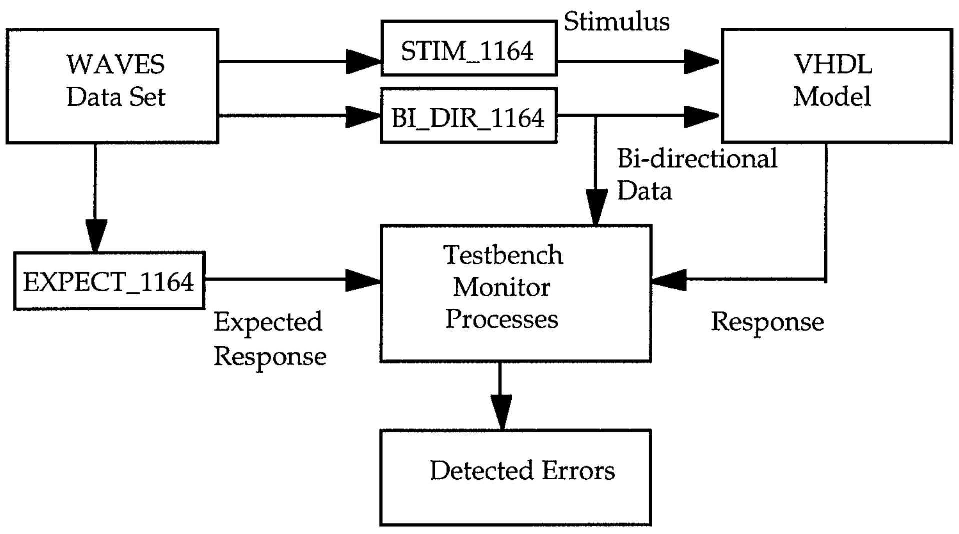


Figure 4.2.1 Bi-Directional Mappings.

The **stim_1164** function converts a WAVES port integer into an 1164 std_logic value. The translation function is overloaded to translate a single WAVES port integer into a std_logic bit or a list of WAVES port integers into a std_logic_vector. The function supports both resolved and unresolved IEEE Std 1164-1993 vector types. Columns 1 and 3 of Table 4.2.1 shows the mapping relationships between the WAVES logic values as described in Chapter 2 and the standard logic simulation code that is produced from the WAVES port integer value. This function, as used in Figure 4.2.1, is used for all model signals that are declared as input only. This function translates the WAVES port integer values into stimulus for input into the model.

The **bi_dir_1164** function also converts a WAVES port integer into an 1164 std_logic value. This function was developed to support the translation of inputs for bi-directional pins. This function translates all of the drive values exactly the same as stim_1164, when the models' port is in the input mode. In order to prevent any conflict when the models' port is in the out state, this function strips off the sense values and replaces them with 'Z', high impedance. This translation of any sense values to high impedance allows the IEEE Std 1164-1993 bus resolution function to resolve the state of the bi-directional signal generated by the model. The translation function is also overloaded to translate a WAVES port integer into a std_logic bit or a list of WAVES port integers into a std_logic_vector. Only resolved std_logic data types are required, since this function was designed for bi-directional signals which require a resolved std_logic data type. Columns 1 and 5 of Table 4.2.1 shows the mapping relationships between the WAVES logic values as described in Chapter 2 and the standard logic simulation code that is produced from the WAVES port integer value.

The **expect_1164** function also converts a single WAVES port integer into an 1164 std_logic value. The difference between expect_1164 and stim_1164 is that the function expect_1164 defines an expected or predicted value of a models' output. This generated signal is the definition of a predicted value which verifies conformance of the models' output. Therefore, for output pins, the expect_1164 functions the same as stim_1164, except the logic value represents the expected response of the model. However, for bi-directional pins, the expect_1164 function strips off the driven values and replaces them with '-', dont_care. This translation of drive values to dont_cares allows the use of this function with any signal specified as OUT or INOUT. The translation function is also overloaded to translate a single port integer into a std_logic bit or a list of WAVES port integers into a std_logic_vector. In this case, the resolved std_logic_vector type is not required. Since this function was designed to translate the logic values for input into a VHDL process, that process requires an unresolved source for explicit definition of the expected models' output. The functionality of the VHDL monitor process that uses the translated value is described in further detail in section 4.3. Columns 1 and 4 of Table 4.2.1 shows the mapping relationships between the WAVES logic values as described in Chapter 2 and the standard logic simulation code which is produced from the WAVES port integer value.

Table 4.2.1 shows the bit translations that will result from a given WAVES port integer and the associated translated function. Column 1 shows the corresponding WAVES logic value as defined in Chapter 2.

4.3 WAVES Compatible Functions

The **compatible** function evaluates the state of two 1164 std_logic bits or std_logic_vectors for compatibility. This function was designed for use in the testbench within a monitor process. This allows the process to be sensitive to the actual signal value from the model and the expected signal value from the WAVES data set. The process utilizing this function determines if the two signals are "compatible" over the simulation period. A simple assertion statement can then be used to notify the designer when the two signals are incompatible. In this function, the actual data value is evaluated to determine if it is compatible to an expected or predicted value. By compatible, we mean: "Is the actual value equivalent to or equal to the value specified by the expected value." The function uses a simple lookup table which contains all of the compatible definitions that were developed. For a more detailed analysis, the complete table is listed in Appendix 3. The order of relationship between the actual and expected values must be preserved, since they carry different meanings. For example, if the actual data is '1' and the expected data was '-' (dont_care), the result would be true. However, if actual data is '-' and the expected data was '1', the result would be false. The function is overloaded to support all of the data types that would be utilized by any design compliant with IEEE Std 1164-1993.

CHAPTER 5: WAVES TESTBENCH TOOL

5.1 Introduction

As mentioned in the previous chapters, prototype tools were developed to aid a user in constructing their WAVES data sets and generating the associated VHDL testbench. This chapter describes the operation of the tools and walks a user through a simple example using a single flip-flop cell as the model design. A brief explanation of the WAVES and VHDL source code generated by the tools will also be given.

5.2 Overview

An X-Windows Graphical User Interface (GUI) was developed using TCL/TK [4] to simplify and enhance the usability of the tool set designed. This GUI provides a single interface for the user to interact with the tool set. For the rest of this chapter, the tool set will be referenced from the main user interface and referred to as XTSTB. Figure 5.2.1 shows the complete VHDL library structure and files supported under this environment. The original WAVES Standard Library is augmented by the WAVES 1164 Library. Together, they establish a WAVES library environment of precompiled functions as discussed in Chapters 2, 3, and 4. The files which the user must develop are located in the library illustrated as "user library." This chapter discusses the generation of the files in this library.

The XTSTB tool set has several features that aid the user when using WAVES and VHDL together. First, the tool generates WAVES packages that are utilized in defining the model specific WAVES data set. The WAVES packages generated by the XTSTB tool are the UUT_pins package, the waveform generator procedure template, and the header file template. Second, the tool will perform a syntax check on the external pattern file that is utilized within the WAVES data set. This syntax check verifies conformance to WAVES level 1 syntax and the std_logic pin codes definitions. Finally, the tool will generate the VHDL testbench which wires the WAVES waveform generator and the model together for design verification. The main input to the tool set is the models' entity declaration. The entity declaration is parsed by the tool set and used to define the appropriate signals and data structures for creation of both the WAVES data set and the VHDL testbench code.

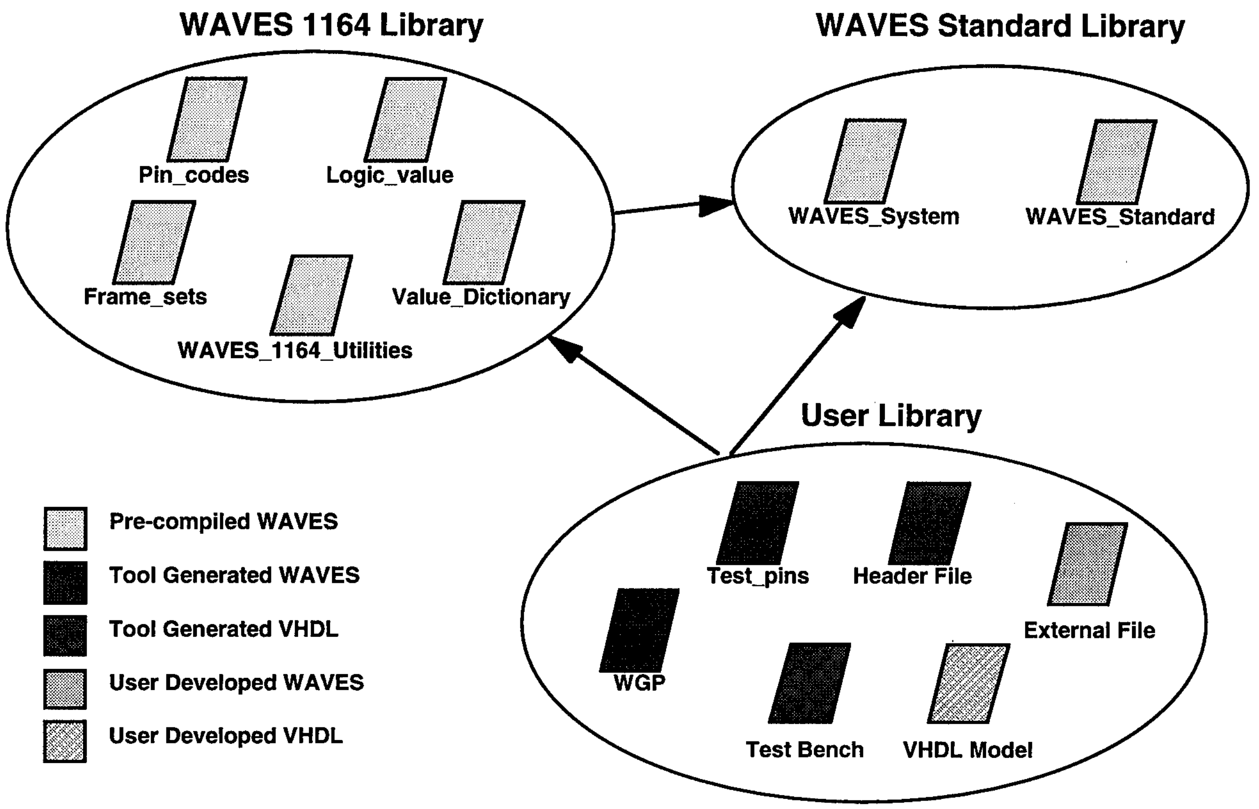


Figure 5.2.1 Final WAVES-VHDL Library Structure.

5.3 SITE SETUP

5.3.1 XTSTB SETUP

Before the XTSTB tool may be used, all required program files must be copied to a common directory where all potential users have read and execute access. This directory must then be added to the search path for execution. The XTSTB GUI interface requires that an environment variable be created which points to the directory path where these files are located. The environment variable command would look something like:

For Unix: → setevn WAVES_GEN /usr/dir_name/.../..

For Windows: → set WAVES_GEN="/usr/dir_name/.../.."

5.3.2 WAVES Libraries Setup

Two libraries must be established within your VHDL environment to support this VHDL-WAVES methodology. These two libraries are named WAVES_STD and WAVES_1164 as shown in Figure 5.2.1. These libraries only need to be created once for any design environment. They contain predefined VHDL and WAVES packages that are referenced by the automatically generated WAVES and testbench files. The WAVES_STD library contains the VHDL implementation of the WAVES standard data types and functions as described in the WAVES Language Reference Manual. The WAVES_1164 library contains the predefined WAVES packages and the testbench utilities packages presented in the earlier chapters of this report. The WAVES files and VHDL testbench generated by the XTSTB tool will utilize these libraries when compiled.

The WAVES standard packages and the WAVES_1164 packages discussed earlier have been merged into two source files to simplify distribution. The files are called "waves_std_lib.vhd" and "waves_1164_lib.vhd" and must be analyzed into the WAVES_STD and WAVES_1164 libraries. The "waves_std_lib.vhd" file contains two packages, WAVES_System and WAVES_Standard. The "waves_1164_lib.vhd" file contains five packages: WAVES_1164_pin_codes, WAVES_1164_logic_values, WAVES_interface, WAVES_1164_frames, and WAVES_1164_utilities.

To establish the site setup, you must first create two libraries within your VHDL environment. These libraries must be named WAVES_STD and WAVES_1164. Then, compile the "waves_std_lib.vhd" file into the WAVES_STD library, and compile the "waves_1164_lib.vhd" file into the WAVES_1164 library. Now you will be able to compile the tool-generated WAVES files and the testbench into a working library for use during simulation.

5.4 D Flip-Flop Example Walk through

The remainder of this chapter will discuss the application of the XTSTB tool set on a D flip-flop. The discussion walks a potential user through the XTSTB tool set process and the WAVES_1164 packages on a step by step basis for the D flip-flop.

5.4.1 Create Working Library

The VHDL code for the D flip-flop is shown in Listing 5.4.1 and is contained in a file named "`d_flip_flop.vhd`." The first step is to create a work library within your VHDL environment and compile the "`d_flip_flop.vhd`" file into that work library.

```
-- behavioral model
-- Positive edge triggered D Flip Flop
library ieee;
use ieee.std_logic_1164.all;
entity d_flip_flop is
  port ( clock : in  std_logic ;
         D     : in  std_logic ;
         Q     : out std_logic ;
         Q_bar : out std_logic );
end d_flip_flop ;

architecture behavioral of d_flip_flop is
begin
main : process ( clock )
  begin
    if clock = '1' then
      Q      <= D ;
      Q_bar  <= not(D) ;
    else
      null;
    end if;
  end process;
end behavioral;
```

Listing 5.4.1 VHDL D-flip-flop Model.

To perform a simulation of the D flip-flop, the WAVES data set must also be created.

5.4.2 Waves Data Set Creation

To create the WAVES data set, the XTSTB interface tool must be started. After the environment variable WAVES_GEN is defined and the directory has been correctly set up, as described in section 5.3, the user simply types "xtstb". The program will display the window shown in Figure 5.4.1. If the WAVES_GEN environment variable is not pointing to the tool directory or the program files are missing, the tool will display an error message requesting the user to correct the WAVES_GEN variable.

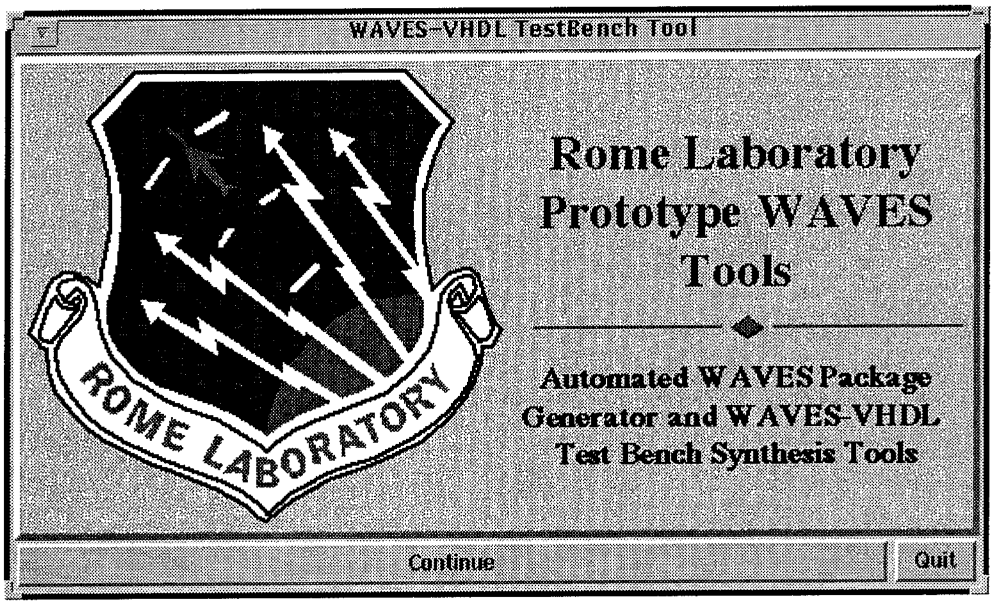
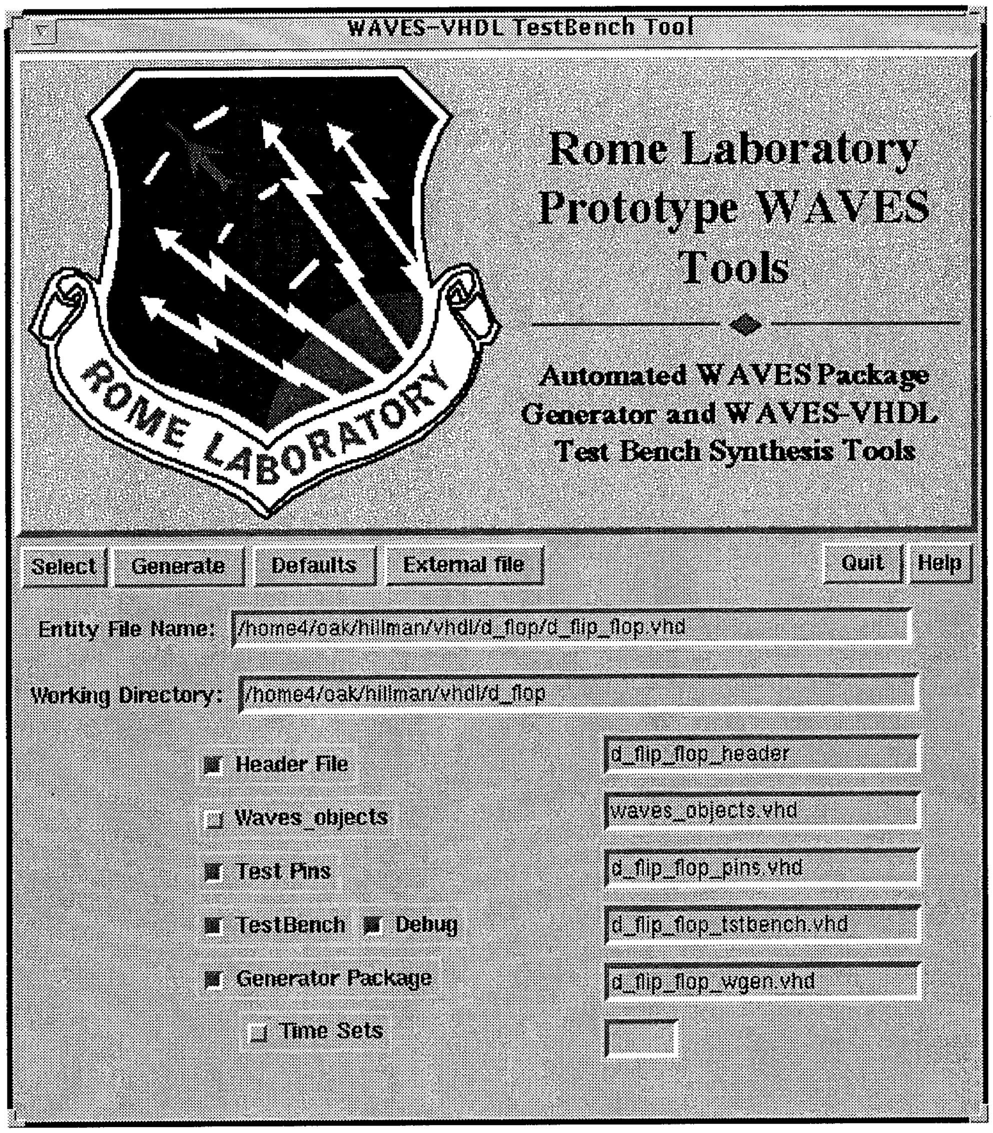


Figure 5.4.1 Initial XTSTB Screen.

Once invoked and active, the user can move the mouse over the continue button and depress the left mouse button. This will bring up the main screen shown in Figure 5.4.2. The user also has the option of quitting the XTSTB tool at this point if desired.

The main display contains a set of selection buttons, checkboxes and entry fields. The buttons each invoke a particular operation. The checkboxes are the optional controls that select the output to be generated.


WAVES-VHDL TestBench Tool
ROME LABORATORY
Rome Laboratory Prototype WAVES Tools
Automated WAVES Package Generator and WAVES-VHDL Test Bench Synthesis Tools
Select
Generate
Defaults
External file
Quit
Help

Entity File Name: /home4/oak/hillman/vhdl/d_flop/d_flip_flop.vhd

Working Directory: /home4/oak/hillman/vhdl/d_flop

■ Header File	d_flip_flop_header
□ Waves_objects	waves_objects.vhd
■ Test Pins	d_flip_flop_pins.vhd
■ TestBench ■ Debug	d_flip_flop_tstbench.vhd
■ Generator Package	d_flip_flop_wgen.vhd
□ Time Sets	

Figure 5.4.2 XTSTB Main Window

5.4.2.1 XTSTB Button Usage

The **Select** button invokes one of two file selector boxes. These interface with the operating system directory structure to select the VHDL file for the model entity or select a working directory for subsequent file generation. The user should select the VHDL input file which contains the VHDL models' design entity, in this case "d_flip_flop.vhd".

The **Generate** button will invoke the tools to generate all of the entries selected by the checkbox options. If an entry data field associated with an active checkbox is not entered correctly, a related error message will be displayed. All files may be generated with a single depression of the

Generate button or they may be generated one at a time.

The **Defaults** button will insert default file names in the entry field for all the output files. If no VHDL entity file has been entered, the default file names generated are prefixed with "waves_". Otherwise, the VHDL models' entity name will be used as the textual prefix value as shown in Figure 5.4.2.

The **Quit** button simply terminates the XTSTB program.

The **Help** button invokes the help menu system which contains text descriptions similar to the descriptions in this chapter.

5.4.2.2 XTSTB Checkbox Usage

The **Header File** checkbox is used to generate the WAVES Header File. You should use this option and generate the header file for configuration control to document the WAVES files. You will need to edit the header file and add any additional information that may be helpful or relevant for use later. This option requires that the user select and enter all of the waves data file name entries.

The WAVES data set **Test Pins** checkbox is used to generate the UUT_pins package that contains the WAVES TEST_PINS declaration. This is the first WAVES package that must be analyzed into the working library. It defines the pin ordering associated with the columns in the external file. This declaration is also used to size all of the waves_objects data structures to match the number of pins in the entity. The WAVES UUT_pins package developed by the tool is generated automatically from the models' entity and requires no user modifications.

The WAVES standard package, WAVES_OBJECTS, is created using the **Waves_objects** checkbox. This is the second file which must be analyzed into the working library. This package must be analyzed *after* the UUT_pins package to allow the compiler to size the waves_objects based on the size of the test_pins declaration. This is the STD WAVES_OBJECTS package and includes all 1164 context clauses plus a clause for the UUT_pins package. This selection was not chosen in this example since this site has a copy of the waves_objects file already available. This file does not change for any WAVES test set that is using the WAVES_1164 library.

The VHDL-WAVES testbench is created using the **Testbench** checkbox. The testbench generated will need little, if any, modification. In this case, the VHDL entity and architecture were located in the same file, therefore, the code generated was complete. You may sometimes need to modify the configuration reference required for your model. The architecture information is extracted, if possible, from the entity declaration if it is contained within the same file. As mentioned above, for the flip flop, no modifications are required. The architecture name "behavioral" was detected and added to the use entity statement.

The **Debug** checkbox option generates a testbench that includes a signal that can be very helpful in debug. If you have a graphical waveform tool you should use the debug option, this signal flags the existence of any failures. This option should probably be a default, since it is such a helpful aid in debugging the model.

Finally, the WAVES data set **Generator Package** checkbox is used to generate the Waveform Generator Procedure. The default procedure is set up to use a single frame set array with a constant period or the period supplied from the external file.

The **Time Sets** checkbox selects a time set style waveform generator using a multiple WAVES timing list. For a complex model where multiple combinations of timing and waveform formats are required, this option would be selected. The numeric entry allowed in the entry field for this option ranges from 1 to n, where n has been set to an arbitrary limit of 20.

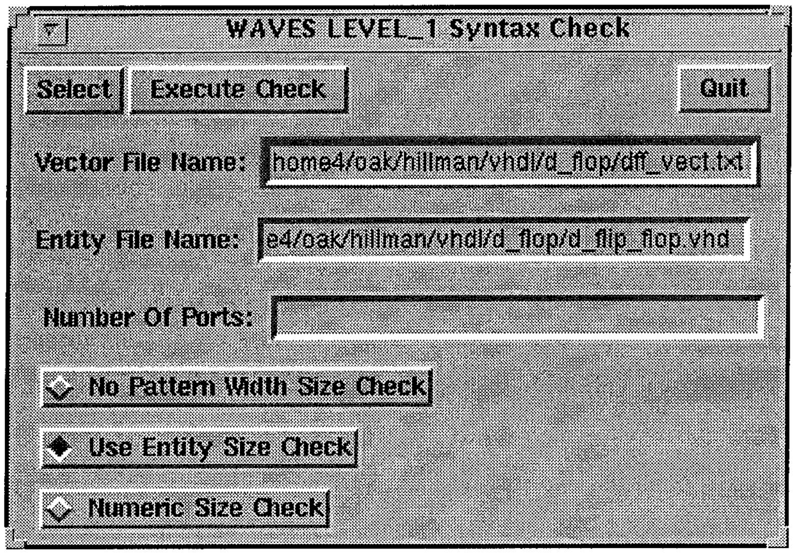
5.4.2.3 Syntax Check Of External Pattern File

Normally you will have to generate the external vector file or use a WAVES level two data set to generate the stimulus. If you use an external file, you should check the syntax. The WAVES internal Read_file_slice function was designed for execution of the WAVES external file and does not gracefully handle all syntax errors. You should use the **External file** button selection to check the syntax of the external file. The flip flop external vector file, "`dff_vect.txt`", is shown in Listing 5.4.2.

```
%clock data  Q  QBAR
    1    1    1   0 : 20 ns;
    0    0    1   0;
    1    0    0   1;
    1    1    1   0;
    1    0    1   1;
```

Listing 5.4.2 External Vector File.

Figure 5.4.3 shows the tool interface when performing the external file syntax check. Since the external file pattern width should usually match the number of individual pins on the entity, the syntax checking also verifies a sizing match between the model and the pattern specified. This check was performed using the entity to define the number of pin elements on the model. The select button was used to open the dialog boxes for file selection. Once the fields being utilized are valid, the **Execute Check** button is used to invoke the syntax check. The radio buttons shown on the bottom of Figure 5.4.3 are used to define the type of size check to perform.



Vector File Name: home4/oak/hillman/vhdl/d_flop/dff_vect.txt

Entity File Name: e4/oak/hillman/vhdl/d_flop/d_flip_flop.vhd

Number Of Ports:

No Pattern Width Size Check [✓ Use Entity Size Check] Numeric Size Check

Figure 5.4.3 Syntax Check Window.

5.4.3 Files Generated

This section contains the outputs generated by the tool set. Any changes that were made to the generated files are shown in boldface to highlight the changes made.

5.4.3.1 Header File

The header file template is generated to provide configuration information. The WAVES source files and order of analysis are automatically captured. All the user has to do is add any site or model specific textual information deemed relevant. The header file should be modified to show any relevant information that may be useful for configuration and permanent reference. The flip-flop header file, "`d_flip_flop_header.txt`", is shown in Listing 5.4.3.

```
-- *************************************************
--
-- ******** Header File for Entity: d_flip_flop
--
-- *************************************************
-- *************************************************
--
-- Data Set Identification Information
--
TITLE         A General Description
DEVICE_ID     d_flip_flop

DATE          Mon Mar 11 14:36:59 1995
ORIGIN        Rome Lab Design Team
AUTHOR        Company or Person
AUTHOR        Maybe Multiple ... Companies or People
DATE          Mon Mar 11 14:36:59 1995
ORIGIN        Modified by Company X Design Team
AUTHOR        Who did it Company or Person

OTHER         Any general comments you want
OTHER         Built Using the WAVES-VHDL 1164 STD Libraries
--
-- Data Set Construction Information
--
WAVES_FILENAME    d_flip_flop_pins.vhd                     WORK
library           WAVES_1164;
use               WAVES_1164.WAVES_1164_Pin_Codes.all;
use               WAVES_1164.WAVES_1164_Logic_Value.all;
use               WAVES_1164.WAVES_Interface.all;
use               WORK.UUT_Test_pins.all;
WAVES_UNIT        WAVES_OBJECTS                            WORK
WAVES_FILENAME    d_flip_flop_wgen.vhd                     WORK
--
EXTERNAL_FILENAME dff_vect.txt               VECTORS
--
WAVEFORM_GENERATOR_PROCEDURE      WORK.waves_d_flip_flop.waveform
```

Listing 5.4.3 WAVES D-Flip Flop Header File.

5.4.3.2 Test Pins Package

The test pins package should never be modified, unless you want to change the pin column references in the external file. The test pins package for our example, "`d_flip_flop_pins.vhd`", is shown in Listing 5.4.4.

```
-- ******** This File Was Automatically Generated  ********
-- ******** By The WAVES-VHDL Tool Set       ********
-- ******** Generated for Entity: d_flip_flop
-- ******** This File Was Generated on: Mon Mar 11 14:36:59 1995
--
--
PACKAGE uut_test_pins IS
TYPE test_pins IS (clock, D, Q, Q_bar);
END uut_test_pins;
```

Listing 5.4.4 WAVES D-Flip Flop Test Pins Package.

5.4.3.3 Waveform Generator Package

Although the waveform generator package generation is automated, you must still edit the file that is produced. The package actually is a template and thus requires modification. The waveform generator procedure template provides a basic structure based on the model that defined pin groupings and signal association for waveform shapes. The tool makes a first cut at establishing all required pin groupings and provides a template section for the user to supply the signal format and timing values associated with the external pattern data. The waveform generator package for the flip-flop example, "`d_flip_flop_wgen.vhd`", is shown in Listing 5.4.5.

The formats pre-assigned and pin groupings created for the model may or may not need to be adjusted. The tool scans for pin names looking for a clock signal named clock or clk so it can assign a pulse format to them. For this model, the tool assigned all pin groups and formats correctly.

The vector file reference generated on line 26 was based on the entity_name with "`_vectors.txt`" appended. Therefore, the file reference was changed to "`dff_vect.txt`", since this is the name of the vector file.

Since the XTSTB tool is unable to predict the time values for the waveform edge placements required for a model, a reasonable limitation, you will *always* have to enter the waveform timing. Default entries are specified only as: `ns`. The values you choose may be entered directly, such as: `10 ns`, or they may be set by using constants such as: `period`, or they may be an equation such as: `period - 3 ns`. For our flip flop example, the pulse_high format on line 33 is changed with times of 10 ns and 20 ns. The non_return format on line 34 for the input pin is modified to be prior to the 10 ns clock, so a value of 5 ns is used. For the expect window any time after 10 ns would be valid, since the model contains no delays. A window time of 13 ns and 20 ns were therefore used on line 35.

```
1  -- ******** This File Was Automatically Generated  ********
2  -- ******** By The WAVES-VHDL Tool Set      ********
3  -- ******** Generated for Entity: d_flip_flop
4  -- ******** This File Was Generated on: Mon Mar 11 14:36:59 1995
5  --
6  --

7  LIBRARY WAVES_STD;
8  USE WAVES_STD.WAVES_Standard.all;

9  USE STD.textio.all;
10 LIBRARY WAVES_1164;
11 USE WAVES_1164.waves_1164_frames.all;
12 USE WAVES_1164.waves_1164_pin_codes.all;
13 USE WAVES_1164.waves_interface.all;
14 USE work.waves_objects.all;
15 USE work.uut_test_Pins.all;

16 PACKAGE WGP_d_flip_flop is
17 PROCEDURE  waveform(SIGNAL WPL : inout WAVES_PORT_LIST);
18 END WGP_d_flip_flop;

19 --------------------------------------------------------

20 PACKAGE BODY WGP_d_flip_flop is

21 -- This is the uut pin declaration pin and ordering
22 -- Remember you need to match the External file to This order
23 --
24 --clock, D, Q, Q_bar

25 PROCEDURE  waveform(SIGNAL WPL : inout WAVES_PORT_LIST) is

26 FILE vector_file : TEXT is in "dff_vect.txt";

27 VARIABLE vector : FILE_SLICE := NEW_FILE_SLICE;

28 -- declare time constants to use or use time literals
29 -- constants or time literals can be used as the frame time values
```

```
CONSTANT outputs: pinset:= new_pinset((Q, Q_bar));

CONSTANT inputs: pinset:= new_pinset((D));


CONSTANT vector_FSA : Frame_set_array :=
New_frame_set_array(Pulse_high(10 ns, 20 ns), clock) +
New_frame_set_array(Non_return(5 ns), inputs) +
New_frame_set_array(window(13 ns, 20 ns), outputs);

VARIABLE timing : time_data := new_time_data(vector_fsa);

BEGIN
loop
READ_FILE_SLICE (vector_file, Vector);  -- get first vector
exit when vector.end_of_file;
apply(wpl, vector.codes.all, Delay(vector.fs_time), timing);
--    or use internal slice time format as below
--    apply(wpl, vector.codes.all, Delay( ns), timing);
end loop;

END waveform;


END WGP_d_flip_flop;
```

Listing 5.4.5 WAVES Waveform Generator Package.

5.4.3.4 WAVES-VHDL TESTBENCH

The testbench generated for this example, "`d_flip_flop_tstbench.vhd`", did not require any modifications as discussed in section 5.4.2.2. It is shown in Listing 5.4.6.

```
-- ******** This File Was Automatically Generated  ********
-- ******** By The WAVES-VHDL Tool Set       ********
-- ******** Generated for Entity: d_flip_flop
-- ******** This File Was Generated on: Mon Mar 11 14:37:02 1995
--
--
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

LIBRARY waves_1164;
USE waves_1164.WAVES_1164_utilities.all;

USE WORK.UUT_test_pins.all;
USE work.waves_objects.all;

USE work.WGP_d_flip_flop.all;
```

```
-- Include component libary references here
-- User Must Modify And ADD component libary references here
-- Include component libary references here

ENTITY test_bench IS
END test_bench;


ARCHITECTURE d_flip_flop_test OF test_bench IS


  --*******************************************************
  --***********CONFIGURATION SPECIFICATION **************
  --*******************************************************

   COMPONENT d_flip_flop
    PORT ( clock               : IN   std_logic;
           D                   : IN   std_logic;
           Q                   : OUT  std_logic;
           Q_bar               : OUT  std_logic);
     END COMPONENT;

 -- Modify entity use statement
 -- User Must Modify modify and declare correct
 --   .. Architecture, Library, Component ..
 -- Modify entity use statement
FOR ALL:d_flip_flop USE ENTITY work.d_flip_flop(behavioral);

  --**********************************************************
  -- stimulus signals for the waveforms mapped into UUT INPUTS
  --**********************************************************

    SIGNAL WAV_STIM_clock                     :std_logic;
    SIGNAL WAV_STIM_D                         :std_logic;

  --****************************************************
  -- Expected signals used in monitoring the UUT OUTPUTS
  --****************************************************

    SIGNAL FAIL_SIGNAL                        :std_logic;
    SIGNAL WAV_EXPECT_Q                       :std_logic;
    SIGNAL WAV_EXPECT_Q_bar                   :std_logic;

  --******************************************************
  -- UUT Output signals used In Monitoring ACTUAL Values
  --******************************************************

    SIGNAL ACTUAL_Q                           :std_logic;
    SIGNAL ACTUAL_Q_bar                       :std_logic;

  --*********************************************************
  -- Bi_directional signals used  for stimulus signals mapped
  -- into UUT INPUTS and also monitoring the UUT OUTPUTS
  --*********************************************************
```

```
--
--
-- No Bidirectional Pins On UUT


  --*************************************************************
  -- WAVES signals OUTPUTing each slice of the waves port list
  --*************************************************************

        SIGNAL wpl  : WAVES_port_list;

BEGIN
  --
  --*************************************************************
  -- process that generates the WAVES waveform
  --*************************************************************

      WAVES: waveform(wpl);

  --*************************************************************
  -- processes that convert the WPL values to 1164 Logic Values
  --*************************************************************

  WAV_STIM_clock                <= STIM_1164(wpl.wpl( 1 ));
  WAV_STIM_D                    <= STIM_1164(wpl.wpl( 2 ));
  WAV_EXPECT_Q                  <= EXPECT_1164(wpl.wpl( 3 ));
  WAV_EXPECT_Q_bar              <= EXPECT_1164(wpl.wpl( 4 ));

  --*****************************************
  -- UUT Port Map - Name Symantics Denote Usage
  --*****************************************

  u1: d_flip_flop
  PORT MAP(
    clock               => WAV_STIM_clock,
    D                   => WAV_STIM_D,
    Q                   => ACTUAL_Q,
    Q_bar               => ACTUAL_Q_bar);


  --*****************************************************
  -- Monitor Processes To Verify The UUT Operational Response
  --*****************************************************

 Monitor_Q:
   PROCESS(ACTUAL_Q, WAV_expect_Q)
   BEGIN
        assert(Compatible (actual => ACTUAL_Q,
                           expected => WAV_expect_Q))
        report "Error on Q output" severity WARNING;

   IF ( Compatible ( ACTUAL_Q,    WAV_expect_Q) ) THEN
     FAIL_SIGNAL <='L'; ELSE FAIL_SIGNAL <='1';
    END IF;
    END PROCESS;


  Monitor_Q_bar:
    PROCESS(ACTUAL_Q_bar, WAV_expect_Q_bar)
    BEGIN
         assert(Compatible (actual => ACTUAL_Q_bar,
                           expected => WAV_expect_Q_bar))
         report "Error on Q_bar output" severity WARNING;

    IF ( Compatible ( ACTUAL_Q_bar,    WAV_expect_Q_bar) ) THEN
      FAIL_SIGNAL <='L'; ELSE FAIL_SIGNAL <='1';
    END IF;
    END PROCESS;


END d_flip_flop_test;
```

Listing 5.4.6 WAVES-VHDL Testbench.

5.4.4 Analyze and Simulate WAVES and VHDL Packages

Once all of the files are generated and modified, the packages may be analyzed and simulated. You would analyze the WAVES data set into the working library created for this model by doing the following:

1. Analyze `d_flip_flop_pins.vhd` into library WORK.
2. Analyze `WAVES_OBJECTS` into library WORK.
3. Analyze `d_flip_flop_wgen.vhd` into library WORK.
4. Analyze `d_flip_flop_tstbench.vhd` into library WORK.

Now you are ready to simulate the flip flop testbench. More detailed model examples are introduced in Chapter 6. Complete listings of these examples are given in the appendices.

CHAPTER 6: EXAMPLES

6.1 INTRODUCTION

This chapter provides three examples to demonstrate the usage of the testbench tools. The first example, an 8-bit parity generator/checker, demonstrates basic usage of the testbench tool. The second example, an 8-bit synchronous up/down counter, demonstrates the testbench tool usage on a model that uses standard logic vectors. The final example, an 8-bit universal shift/storage register, demonstrates testbench tool usage on a bi-directional I/O model. The examples were successfully simulated using three different VHDL simulators on two different hardware platforms, thus also demonstrating the portability of VHDL and WAVES.

6.2 EXAMPLE 1: 54/74180: 8 - BIT PARITY GENERATOR/CHECKER

This example demonstrates usage of the testbench tool for generating a VHDL/WAVES testbench with a single output monitor process. The external file consists only of pin codes. All timing is performed in the WAVES generator. This example was generated using the command line interface to the tool set. The individual commands used to invoke the tools are specified.

6.2.1 DEVICE SPECIFICATIONS

This parity checker was modeled after the 54/74180 series taken from the Fairchild TTL Data Book printed in December 1978. The '180 is a monolithic 8-bit parity checker/generator which features control inputs and even/odd outputs to enhance operation in either odd or even parity applications. Cascading these circuits allows unlimited word length expansion. A typical application would be to generate and check parity on data being transmitted from one register to another. It is a 14 pin device with 8 data inputs (I_0 through I_7), 2 control inputs (Odd input and Even Input) and 2 outputs (Odd parity Output and Even Parity Output). Table 6.1.1 lists the pin numbers, pin name and a short description of each pin. Table 6.1.2 is the truth table for the device.

Pin Number	Pin Name	Description
1	I6	Data Input 6
2	I7	Data Input 7
3	EI	Even Input
4	OI	Odd Input
5	SE	Even Parity Output
6	SO	Odd Parity Output
7	GND	Ground
8	I0	Data Input 0
9	I1	Data Input 1
10	I2	Data Input 2
11	I3	Data Input 3
12	I4	Data Input 4
13	I5	Data Input 5
14	Vcc	Supply Voltage

Table 6.1.1 8-Bit Parity Checker/Generator Pin Labels.

Inputs			Outputs	
Parity	Even	Odd	Even	Odd
Even	H	L	H	L
Odd	H	L	L	H
Even	L	H	L	H
Odd	L	H	H	L
X	H	H	L	L
X	L	L	H	H

Table 6.1.2 8-Bit Parity Checker/Generator Truth Table.

6.2.2 TOOL USAGE EXPLANATION

As explained in previous chapters, once the entity architecture is completed, the necessary WAVES files needed to test are the pins file, header file, objects file, generator file, testbench file and the vector file. The process by which these files are generated for this example are discussed below.

First, the testbench tool is invoked by:

```
    wfgen -p pg_e_a.vhd > pg_pins.vhd
```

This will automatically generate the necessary pins file.

Second, the tool is invoked by:

```
    wfgen -h -p:pg_pins.vhd -g pg_gen pg_e_a.vhd > pg_head.vhd
```

This will produce the header file. This file may now be edited for any additional information.

Third, the tool is invoked by:

```
        wfgen -o pg_e_a.vhd > pg_objects.vhd
```

This will automatically generate the necessary WAVES object file.

Fourth, the tool is invoked by:

```
        wfgen -g pg_e_a.vhd > pg_gen.vhd
```

This will produce the WAVES waveform generator package. A portion of this file is shown in Listing 6.2.1 below.

```
                    :
                    :
 FILE vector_file   : TEXT IS IN "pg_e_a_vectors.txt";
                    :
                    :
  CONSTANT vector_fsa : frame_set_array :=
      new_frame_set_array(pulse_high( ns, ns), a_clock) +
      new_frame_set_array(non_return( ns), inputs) +
      new_frame_set_array(window( ns, ns), outputs);
                    :
                    :
```

Listing 6.2.1 Unedited waveform generator frame set array declaration.

This file must now be edited. First, the vector file name is edited. Second, the timing information must be supplied. Finally, since no clock exists for this model, this line may be commented out or deleted, as shown below.

```
                    :
                    :
 FILE vector_file   : TEXT IS IN "vectors.txt";
                    :
                    :
  CONSTANT vector_fsa : frame_set_array :=
      new_frame_set_array(non_return( 0 ns), inputs) +
      new_frame_set_array(window( 70 ns, 100 ns), outputs);
                    :
                    :
```

Listing 6.2.2 Edited waveform generator frame set array declaration.

Finally, the tool is invoked by:

```
        wtstb -t pg_e_a.vhd > pg_test.vhd
```

This will automatically generate the necessary testbench file.

A syntax check of the pattern file is performed by:

```
    wextern -E:pg_e_a.vhd pg_e_a_vectors.txt
```

The necessary WAVES files to test and simulate this model have now been written with minimal effort by the user.

6.2.3 Simulation

All of the WAVES files are then compiled into the working directory and the simulation was then performed. For this example, the Model Technology simulator running on a Pentium 100 based PC running Win95 was used.

6.3 Example 2: SN54/74ALS8169: Synchronous 8 - Bit Up/Down Binary Counter

This VHDL model is slightly more complex than the parity generator model in example 1, as it contains IEEE 1164 Standard Logic Vectors as an input and an output. This is handled easily by the testbench tool as seen in the testbench itself for the Up Down Counter (Appendix 6). The tool creates actual and expected Standard Logic Vectors for the Q outputs and a WAVES Stimulus Standard Logic Vector for the eight bit input. Additionally, internal signal assignment statements are generated for converting the WAVES Port List (WPL) values to the 1164 Logic Values that will be needed for stimulus and comparison of model responses. Finally, an 1164 Standard Logic Vector Monitor process is created for monitoring the outputs from the VHDL model.

6.3.1 Device Specifications

The Texas Instruments SN54ALS8169 - Synchronous 8-Bit Up/Down Binary Counter is a 24 pin device with 8 data inputs (A through H), 8 data outputs (Qa through Qh), and 5 control inputs (Load Bar, U/D Bar, ENT Bar, RCO Bar and ENP Bar). The device is easily cascadable for n-bit synchronous applications and fully programmable (may be preset to any number between 0 and 255). A complete description of this part may be obtained from the Texas Instruments Data Book, 1987.

Table 6.2.1 lists the Up/Down Counter's pin numbers, names and a short description of their functionality. Table 6.2.2 is the truth table of this device (only four bits are shown to conserve space).

Pin Number	Pin Name	Description
1	Load Bar	Load Binary (Preset)
2	U/D Bar	Count Up when high, Count Down when low
3	A	Data Input A (LSB)
4	B	Data Input B
5	C	Data Input C
6	D	Data Input D
7	E	Data Input E
8	F	Data Input F
9	G	Data Input G
10	H	Data Input H (MSB)
11	ENT Bar	Enable Counting, also fed forward for ripple carry out
12	GND	Ground
13	RCO Bar	Ripple Carry Out
14	CLK	Clock
15	Qh	Result Output H (MSB)
16	Qg	Result Output G
17	Qf	Result Output F
18	Qe	Result Output E
19	Qd	Result Output D
20	Qc	Result Output C
21	Qb	Result Output B
22	Qa	Result Output A (LSB)
23	ENP Bar	Enable Counting
24	Vcc	Supply Voltage

Table 6.2.1 Synchronous 8-Bit Up/Down Binary Counter Pin Labels.

Operation	Inputs at time t_n									Outputs at time t_{n-1}				
Load	Clock	ENP	ENT	LOAD	A	B	C	D	U/D	Q_a	Q_b	Q_c	Q_d	RCO
	↑	L	L	L	X	X	X	X	X	A	B	C	D	L if count =15 H if count ≠15
Count Up	↑	L	L	H	X	X	X	X	H	Previous count plus 1				L if count =15 H if count ≠15
Count Down	↑	L	L	H	X	X	X	X	L	Previous count minus 1				L if count =15 H if count ≠15
Inhibit	↑	H	L	H	X	X	X	X	X	No change				No change
	↑	L	H	H	X	X	X	X	X					L
	↑	H	H	H	X	X	X	X	X					L

Table 6.2.2 Synchronous 8-Bit Up/Down Binary Counter Truth Table.

The external file used for testing the Up/Down Counter VHDL model consists of 6 different tests. The first test simply loads in the value of zero and counts up to seventy nine, checking if the model counts up. The second test loads in 250 and counts up through 255 to 16 to check if RCO_Bar goes low at 255. The third test loads in 100 and counts down through 0 to 245, checking to see if the model counts down and if RCO_bar goes low at 0 while counting down. The fourth test loads in 0, counts up to 2, disables counting by resetting both ENT_Bar and ENP_Bar, and then resetting both again and counts back down through 0 to 253. Tests 5 and 6 are similar except that they simply disable counting by resetting only ENT_Bar and the only ENP_Bar respectively.

6.3.2 Tool Usage Explanation

Refer to Appendix 6 for the complete file listings.

Changes were made to the testbench tool generated waveform generator procedure:

1) The external file name had to be changed to the proper name on line 37 from:

```
FILE vector_file : TEXT IS IN "eight_bit_sync_ud_cntr_vectors.txt";
```

to:

```
FILE vector_file : TEXT IS IN "synctr_vectors.txt";
```

2) The in_pins constant, shown in Listing 6.3.1, was modified because `ent_bar` and `enp_bar` have different timing than `load_bar` and `up_down_bar`. Therefore, a new constant named `enable_pins` was created, shown in Listing 6.3.2:

```
CONSTANT in_pins: pinset:= new_pinset((load_bar,up_down_bar,
                                       ent_bar, enp_bar));
```

Listing 6.3.1 Unedited in_pins constant declaration.

to:

```
CONSTANT in_pins: pinset:= new_pinset((load_bar,up_down_bar));
CONSTANT enable_pins : pinset := new_pinset((ent_bar, enp_bar));
```

Listing 6.3.2 Edited in_pins constant declaration.

3) The code grouping beginning at line 62 shown in Listing 6.3.3, had to be modified to insert a new frame array for the constant `enable_pins`. Also, the absolute timing for the clock, inputs and outputs was added as shown in Lisitng 6.3.4:

```
CONSTANT vector_fsa : frame_set_array :=
    new_frame_set_array(pulse_high( ns, ns), clk) +
    new_frame_set_array(non_return(ns), inputs) +
    new_frame_set_array(window(ns, ns), outputs);
```

Listing 6.3.3 Unedited waveform generator frame set array declaration.

to:

```
CONSTANT vector_fsa : frame_set_array :=
    new_frame_set_array(pulse_high(15 ns, 30 ns), clk) +
    new_frame_set_array(non_return(20 ns), enable_pins) +
    new_frame_set_array(non_return(0 ns), inputs) +
    new_frame_set_array(window(20 ns, 27 ns), outputs);
```

Listing 6.3.4 Edited waveform generator frame set array declaration.

6.3.3 Simulation

The Cadence Leapfrog simulator running on a Sun SPARC IPX running SunOS was used for this model verification.

6.4 Example 3: 54LS/74LS299: 8 - Input Universal Shift/Storage Register

This VHDL model demonstrates how the testbench generation process is performed when the model contains bi-directional pins. The testbench generation tool created the complementary WAVES data files and the VHDL testbench. The generation tool created two time sets in the WAVES waveform generator file requiring that the external file specify which time set was to be used. Two time sets were needed to specify one set of conditions when the bi-directional pins were operating as input pins and the second time set to identify the output characteristics.

6.4.1 Device Specifications

This register was modeled after the 54LS/74LS299 series taken from the Fairchild TTL Data Book printed in December of 1978. The 54LS/74LS299 8-input universal shift/storage register with common parallel input/output pins, is an 8-bit universal shift/storage register with 3-state outputs. Four modes of operation are possible: hold(store), shift left, shift right, and load data. The parallel load inputs and flip-flop outputs are multiplexed to reduce the total number of package pins.

Separate outputs are provided for flip-flops Q0 and Q7 to allow easy cascading. A separate active LOW Master Reset is used to reset the register. The VHDL model generated is a dataflow description of the part.

Pin Number	Model Name	Pin Name	Description
1	selection(0)	S0	Mode Selection Input 0
2	enable_out(0)	NOT OE1	3-State Output Enable Input 1
3	enable_out(1)	NOT OE2	3-State Output Enable Input 2
4	IO(6)	IO6	Parallel Data Input/ 3-State Parallel Output 6
5	IO(4)	IO4	Parallel Data Input/ 3-State Parallel Output 4
6	IO(2)	IO2	Parallel Data Input/ 3-State Parallel Output 2
7	IO(0)	IO0	Parallel Data Input/ 3-State Parallel Output 0
8	OUT_0	Q0	Serial Output 0
9	Master_Reset	NOT MR	Asynchronous Master Reset Input (Active LOW)
10	--------	GND	Ground
11	Data_0	DS0	Serial Data Input for Right Shift
12	Clock	CP	Clock
13	IO(1)	IO1	Parallel Data Input/ 3-State Parallel Output 1
14	IO(3)	IO3	Parallel Data Input/ 3-State Parallel Output 3
15	IO(5)	IO5	Parallel Data Input/ 3-State Parallel Output 5
16	IO(7)	IO7	Parallel Data Input/ 3-State Parallel Output 7
17	OUT_7	Q7	Serial Output 7
18	Data_7	DS7	Serial Data Input for Left Shift
19	selection(1)	S1	Mode Selection Input 1
20	---------	VCC	SUPPLY VOLTAGE

Table 6.3.1 8-Input Universal Shift/Storage Register Pin Labels.

Table 6.3.1 lists the register's pin numbers, model and pin names and a short description of each pins' functionality. The truth table for this device is shown in Table 6.3.2.

INPUTS				RESPONSE
MR	S_1	S_0	CP	
L	X	X	X	Asynchronous Reset: Q_0 - Q_7 = LOW
H	H	H	$\uparrow$	Parallel Load: $I/O_n \rightarrow Q_n$
H	L	H	$\uparrow$	Shift Right: $Ds_0 \rightarrow Q_0$, $Q_0 \rightarrow Q_1$, etc.
H	H	L	$\uparrow$	Shift Left: $Ds_7 \rightarrow Q_7$, $Q_7 \rightarrow Q_6$, etc.
H	L	L	X	Hold

Table 6.3.2 8-Input Universal Shift/Storage Register Truth Table.

The type of operation to be performed is determined by the S0 and S1 selection pins and shown in Table 6.3.2. All flip-flop outputs are brought out through tri-state buffers to separate I/O pins that also serve as data inputs in the parallel load mode. Q0 and Q7 are also brought out on other pins for expansion in serial shifting of longer words. A LOW signal on the master reset pin overrides the select and clock pulse inputs and resets the flip-flops. All other state changes are initiated by the rising edge of the clock. Inputs can change when the clock is in either state provided that the recommended setup and hold times, relative to the rising edge of the cock pulse, are observed. A high signal on either of the output enable pins disables the tri-state buffers and puts the I/O pins in the high impedance state. In this condition, the shift, hold, load and reset operations can still occur. The tri-state buffers are also disabled by high signals on both S0 and S1 in preparation for a parallel load operation.

The external file used for testing the LS299 tested all five of its operations; reset, shift right, shift left, hold and load. The external file format, shown in Appendix A7.6, lists the conditions on the pins (selection, output enable, clock, data serial 0, data serial 7, master reset, and the I/O pins, from left to right) followed by the identifier for the proper time set. Time set one sets the I/O pins to be compared (output mode) and time set two drives the I/O as input. The first vector resets the register, then six shift right operations are performed followed by five shift left operations. The resulting output is then held for five cycles. The register is then loaded with data and the output disabled with the loaded information shifted left. The I/O pins are tested for a tri-state condition and the last operation enables the outputs and performs a final shift left operation.

6.4.2 Tool Usage Explanation

Appendix 7 contains a complete listing of all of the files, both tool generated and hand created, for this model. All of the generated files were produced by the WAVES testbench generation tool as described in Chapter 5.

The only changes which were required of the tool generated files were to the waveform generator procedure. The two changes which were necessary to make are outlined below:

1) The external file name had to be changed to the proper name:

```
FILE vector_file : TEXT IS IN "shift_register_vectors.txt";
```

to:

```
FILE vector_file : TEXT IS IN "vectors.txt";
```

2) The code grouping, which is shown in Listing 6.4.1, had to be modified in two ways. First, for both frame set 1 and 2, a choice of the compare or drive format had to be made. This involves uncommenting the appropriate line for each frame set. Second, the absolute timing for the clock,

inputs and outputs had to be added. The changed listing is shown in Lisitng 6.4.2.

```
          --
          -- Frame Set 1
          --
          (  delay   => delay ( ns ),
             timing => new_time_data(
              new_frame_set_array(pulse_high( ns, ns), clock) +
    -- select compare            New_frame_set_array(Window( ns, ns), IO) +
    -- or drive format           New_frame_set_array(Non_return( ns), IO) +
              new_frame_set_array(non_return( ns), inputs) +
              new_frame_set_array(window( ns, ns), outputs)
          )),
          --
          -- Frame Set 2
          --
          (  delay   => delay ( ns ),
             timing => new_time_data(
              new_frame_set_array(pulse_high( ns, ns), clock) +
    -- select compare            New_frame_set_array(Window( ns, ns), IO) +
    -- or drive format           New_frame_set_array(Non_return( ns), IO) +
              new_frame_set_array(non_return( ns), inputs) +
              new_frame_set_array(window( ns, ns), outputs)
          ) ) );
```

Listing 6.4.1 Unedited waveform generator frame set declaration.

to:

```
          --
          -- Frame Set 1
          --
          (  delay   => delay ( 100 ns ),
             timing => new_time_data(
              new_frame_set_array(pulse_high( 50 ns, 80 ns), clock) +
              new_frame_set_array(window( 85 ns, 95 ns), io) +
    -- or drive format           New_frame_set_array(Non_return( ns), IO) +
              new_frame_set_array(non_return( 5 ns), inputs) +
              new_frame_set_array(window( 85 ns, 95 ns), outputs)
          )),
          --
          -- Frame Set 2
          --
          (  delay   => delay ( 100 ns ),
             timing => new_time_data(
              new_frame_set_array(pulse_high( 50 ns, 80 ns), clock) +
    -- select compare            New_frame_set_array(Window( ns, ns), IO) +
              new_frame_set_array(non_return( 5 ns), io) +
              new_frame_set_array(non_return( 5 ns), inputs) +
              new_frame_set_array(window( 85 ns, 95 ns), outputs)
          ) ) );
```

Listing 6.4.2 Edited waveform generator frame set declaration.

6.4.3 Simulation

This model was verified using the Synopsys simulator on a Sun SPARC IPX running SunOS.

REFERENCES

1. Institute of Electrical and Electronics Engineers: IEEE Standard for Waveform and Vector Exchange (WAVES). IEEE Std 1029.1-1992. September 28, 1992; The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017-2394, USA. ISBN 1-55937-195-1.

2. Institute of Electrical and Electronics Engineers: IEEE Standard VHDL Language Reference Manual. ANSI/IEEE Std 1076-1993. June 4, 1994; The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017-2394, USA. ISBN 1-55937-376-8.

3. Institute of Electrical and Electronics Engineers: IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164). IEEE Std 1164-1993. May 26, 1993; The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017-2394, USA. ISBN 1-55937-299-0.

4. Ousterhout, John K.: Tcl and the Tk Toolkit. Addison-Wesley Publishing Company; 1994. ISBN 0-201-63337-X.

5. Flynn, C.J., Hall, F.G., Hanna, J.P., Nassif, M.P., and Pronobis, M.T., "Top-Down Design Through Test using VHDL and WAVES," Proceedings 1995 International Conference on Electronic Hardware Description Languages (ICEHDL), January 1995, pp. 21-25.

6. Flynn, C.J., Hall, F.G, Hanna, J.P., and Pronobis, M.T., "Using WAVES in a Top-Down Design Methodology," Proceedings VHDL International Users' Forum, Fall Conference, November 1994, pp.12.1-12.6.

7. Pronobis, M., Hillman, R. and Flynn, C., "Test Insertion Without Being a Test Expert," Proceedings VHDL International Users' Forum, Fall Conference, October 1995, pp. 10.1-10.8.

APPENDIX 1: WAVES_1164 PACKAGES

A1.1 WAVES_1164_PIN_CODES PACKAGE

```
PACKAGE waves_1164_pin_codes IS
    CONSTANT pin_codes : STRING := "X01ZWLH-";
END waves_1164_pin_codes;
```

A1.2 WAVES_1164_LOGIC_VALUE PACKAGE

```
LIBRARY waves_std;
USE waves_std.waves_standard.ALL;
PACKAGE waves_1164_logic_value IS

    TYPE logic_value IS
                 (
                   dont_care,
                   sense_x,
                   sense_0,
                   sense_1,
                   sense_z,
                   sense_w,
                   sense_l,
                   sense_h,
                   drive_x,
                   drive_0,
                   drive_1,
                   drive_z,
                   drive_w,
                   drive_l,
                   drive_h );

FUNCTION value_dictionary( value : logic_value )
       RETURN event_value;

END waves_1164_logic_value;


PACKAGE BODY waves_1164_logic_value IS

  FUNCTION value_dictionary( value : logic_value )
         RETURN event_value IS

  BEGIN
    CASE value IS
      WHEN drive_x =>
        RETURN state     = unknown AND
               strength  = drive AND
               direction = stimulus;
      WHEN drive_0 =>
        RETURN state     = low AND
               strength  = drive AND
               direction = stimulus;
      WHEN drive_1 =>
        RETURN state     = high AND
               strength  = drive AND
               direction = stimulus;
      WHEN drive_z =>
        RETURN state     = midband AND
               strength  = disconnected AND
               direction = stimulus;
      WHEN drive_w =>
        RETURN state     = unknown AND
               strength  = resistive AND
               direction = stimulus;
      WHEN drive_l =>
        RETURN state     = low AND
               strength  = resistive AND
               direction = stimulus;
      WHEN drive_h =>
        RETURN state     = high AND
               strength  = resistive AND
               direction = stimulus;
      WHEN sense_x =>
        RETURN state     = unknown AND
               strength  = drive AND
               direction = response AND
               relevance = required;
      WHEN sense_0 =>
        RETURN state      = low AND
               strength   = drive AND
               direction = response AND
               relevance = required;
      WHEN sense_1 =>
        RETURN state      = high AND
               strength   = drive AND
               direction = response AND
               relevance = required;
      WHEN sense_z =>
        RETURN state      = midband AND
               strength   = disconnected AND
               direction = response AND
               relevance = required;
      WHEN sense_w =>
        RETURN state      = unknown AND
               strength   = resistive AND
               direction = response AND
               relevance = required;
      WHEN sense_l =>
        RETURN state      = low AND
               strength   = resistive AND
               direction = response AND
               relevance = required;
      WHEN sense_h =>
        RETURN state      = high AND
               strength   = resistive AND
               direction = response AND
               relevance = required;
      WHEN dont_care =>
        RETURN unspecified;
    END CASE;
END value_dictionary;

END waves_1164_logic_value;
```

APPENDIX 2: WAVES_1164_FRAMES PACKAGE

```
LIBRARY waves_1164;
USE waves_1164.waves_1164_pin_codes.ALL;
USE waves_1164.waves_1164_logic_value.ALL;
USE waves_1164.waves_interface.ALL;
PACKAGE waves_1164_frames IS

  --
  -- Declare functions that return Frame Sets.
  --

  FUNCTION non_return( t1 : TIME ) RETURN frame_set;
  FUNCTION return_low( t1, t2 : TIME ) RETURN frame_set;
  FUNCTION return_high( t1, t2 : TIME ) RETURN frame_set;
  FUNCTION surround_complement( t1, t2 : TIME ) RETURN frame_set;
  FUNCTION pulse_low( t1, t2 : TIME ) RETURN frame_set;
  FUNCTION pulse_low_skew( t0, t1, t2 : TIME ) RETURN frame_set;
  FUNCTION pulse_high( t1, t2 : TIME ) RETURN frame_set;
  FUNCTION pulse_high_skew( t0, t1, t2 : TIME ) RETURN frame_set;
  FUNCTION window( t1, t2 : TIME ) RETURN frame_set;
  FUNCTION window_skew( t0, t1, t2 : TIME ) RETURN frame_set;

END waves_1164_frames;

PACKAGE BODY waves_1164_frames IS

  --
  -- Frame Set function definitions.
  --

  FUNCTION non_return( t1 : TIME ) RETURN frame_set IS

    CONSTANT edge : event_time := etime( t1 );

  BEGIN
    RETURN
      new_frame_set( 'X', frame_event( (drive_X, edge) ) ) +
      new_frame_set( '0', frame_event( (drive_0, edge) ) ) +
      new_frame_set( '1', frame_event( (drive_1, edge) ) ) +
      new_frame_set( 'Z', frame_event( (drive_Z, edge) ) ) +
      new_frame_set( 'W', frame_event( (drive_W, edge) ) ) +
      new_frame_set( 'L', frame_event( (drive_L, edge) ) ) +
      new_frame_set( 'H', frame_event( (drive_H, edge) ) ) +
      new_frame_set( '-', frame_event );
  END non_return;

  FUNCTION return_low( t1, t2 : TIME ) RETURN frame_set IS

    CONSTANT edge1 : event_time := etime( t1 );
    CONSTANT edge2 : event_time := etime( t2 );

  BEGIN
    ASSERT t1 < t2
    REPORT "Timing violation in Return_Low frames." &
           "The inequality : T1 < T2 Must hold."
    SEVERITY FAILURE;
    RETURN
      new_frame_set( 'X', frame_elist( ((drive_X, edge1),
                                        (drive_0, edge2)) ) ) +
      new_frame_set( '0', frame_event( ( drive_0, edge1)  ) ) +
      new_frame_set( '1', frame_elist( ((drive_1, edge1),
                                        (drive_0, edge2)) ) ) +
      new_frame_set( 'Z', frame_elist( ((drive_Z, edge1),
                                        (drive_0, edge2)) ) ) +
      new_frame_set( 'W', frame_elist( ((drive_W, edge1),
                                        (drive_0, edge2)) ) ) +
      new_frame_set( 'L', frame_elist( ((drive_L, edge1),
                                        (drive_0, edge2)) ) ) +
      new_frame_set( 'H', frame_elist( ((drive_H, edge1),
                                        (drive_0, edge2)) ) ) +
      new_frame_set( '-', frame_event( ( drive_0, edge2 ) ) );
```

```
END return_low;


FUNCTION return_high( t1, t2 : TIME ) RETURN frame_set IS

  CONSTANT edge1 : event_time := etime( t1 );
  CONSTANT edge2 : event_time := etime( t2 );

BEGIN
  ASSERT t1 < t2
  REPORT "Timing violation in Return_High frames." &
         "The inequality: T1 < T2 Must hold."
  SEVERITY FAILURE;
  RETURN
    new_frame_set( 'X', frame_elist( ((drive_X, edge1),
                                      (drive_1, edge2)) ) ) +
    new_frame_set( '0', frame_elist( ((drive_0, edge1),
                                      (drive_1, edge2)) ) ) +
    new_frame_set( '1', frame_event( ( drive_1, edge1)  ) ) +
    new_frame_set( 'Z', frame_elist( ((drive_Z, edge1),
                                      (drive_1, edge2)) ) ) +
    new_frame_set( 'W', frame_elist( ((drive_W, edge1),
                                      (drive_1, edge2)) ) ) +
    new_frame_set( 'L', frame_elist( ((drive_L, edge1),
                                      (drive_1, edge2)) ) ) +
    new_frame_set( 'H', frame_elist( ((drive_H, edge1),
                                      (drive_1, edge2)) ) ) +
    new_frame_set( '-', frame_event( ( drive_1, edge2 ) ) );
END return_high;


FUNCTION surround_complement( t1, t2 : TIME) RETURN frame_set IS

  CONSTANT edge0 : event_time := etime( 0 ns );
  CONSTANT edge1 : event_time := etime( t1 );
  CONSTANT edge2 : event_time := etime( t2 );

BEGIN
  ASSERT t1 < t2
  REPORT "Timing violation in Surround_Complement frames.  " &
         "The inequality: T1 < T2 Must hold."
  SEVERITY FAILURE;
  RETURN
    new_frame_set( 'X', frame_event( ( drive_X, edge1)  ) ) +
    new_frame_set( '0', frame_elist( ((drive_1, edge0),
                                      (drive_0, edge1),
                                      (drive_1, edge2)) ) ) +
    new_frame_set( '1', frame_elist( ((drive_0, edge0),
                                      (drive_1, edge1),
                                      (drive_0, edge2)) ) ) +
    new_frame_set( 'Z', frame_event( ( drive_Z, edge1)  ) ) +
    new_frame_set( 'W', frame_event( ( drive_W, edge1)  ) ) +
    new_frame_set( 'L', frame_elist( ((drive_H, edge0),
                                      (drive_l, edge1),
                                      (drive_H, edge2)) ) ) +
    new_frame_set( 'H', frame_elist( ((drive_L, edge0),
                                      (drive_h, edge1),
                                      (drive_L, edge2)) ) ) +
    new_frame_set( '-', frame_event );
END surround_complement;


FUNCTION pulse_low( t1, t2 : TIME ) RETURN frame_set IS

  CONSTANT edge0 : event_time := etime( 0 ns );
  CONSTANT edge1 : event_time := etime( t1 );
  CONSTANT edge2 : event_time := etime( t2 );

BEGIN
  ASSERT t1 < t2
  REPORT "Timing violation in Pulse_Low frames." &
         "The inequality: T1 < T2 Must hold."
  SEVERITY FAILURE;
  RETURN
    new_frame_set( 'X', frame_event ) +
    new_frame_set( '0', frame_elist( ((drive_1, edge0),
                                      (drive_0, edge1),
                                      (drive_1, edge2)) ) ) +
    new_frame_set( '1', frame_event( ( drive_1, edge0)  ) ) +
    new_frame_set( 'Z', frame_event ) +
    new_frame_set( 'W', frame_event ) +
     new_frame_set( 'L', frame_elist( ((drive_H, edge0),
                                        (drive_L, edge1),
                                        (drive_H, edge2)) ) ) +
     new_frame_set( 'H', frame_event( ( drive_H, edge0)   ) ) +
     new_frame_set( '-', frame_event );
END pulse_low;


FUNCTION pulse_low_skew( t0, t1, t2 : TIME ) RETURN frame_set IS

   CONSTANT edge0 : event_time := etime( t0 );
   CONSTANT edge1 : event_time := etime( t1 );
   CONSTANT edge2 : event_time := etime( t2 );

BEGIN
   ASSERT t0 < t1 AND t1 < t2
   REPORT "Timing violation in Pulse_Low frames." &
          "The inequality: T0 < T1 < T2 Must hold."
   SEVERITY FAILURE;
   RETURN
     new_frame_set( 'X', frame_event ) +
     new_frame_set( '0', frame_elist( ((drive_1, edge0),
                                        (drive_0, edge1),
                                        (drive_1, edge2)) ) ) +
     new_frame_set( '1', frame_event( ( drive_1, edge0)   ) ) +
     new_frame_set( 'Z', frame_event ) +
     new_frame_set( 'W', frame_event ) +
     new_frame_set( 'L', frame_elist( ((drive_H, edge0),
                                        (drive_L, edge1),
                                        (drive_H, edge2)) ) ) +
     new_frame_set( 'H', frame_event( ( drive_H, edge0)   ) ) +
     new_frame_set( '-', frame_event );
END pulse_low_skew;


FUNCTION pulse_high( t1, t2 : TIME ) RETURN frame_set IS

   CONSTANT edge0 : event_time := etime( 0 ns );
   CONSTANT edge1 : event_time := etime( t1 );
   CONSTANT edge2 : event_time := etime( t2 );

BEGIN
   ASSERT t1 < t2
   REPORT "Timing violation in Pulse_High frames." &
          "The inequality: T1 < T2 Must hold."
   SEVERITY FAILURE;
   RETURN
     new_frame_set( 'X', frame_event ) +
     new_frame_set( '0', frame_event( ( drive_0, edge0)   ) ) +
     new_frame_set( '1', frame_elist( ((drive_0, edge0),
                                        (drive_1, edge1),
                                        (drive_0, edge2)) ) ) +
     new_frame_set( 'Z', frame_event ) +
     new_frame_set( 'W', frame_event ) +
     new_frame_set( 'L', frame_event( ( drive_L, edge0)   ) ) +
     new_frame_set( 'H', frame_elist( ((drive_L, edge0),
                                        (drive_H, edge1),
                                        (drive_L, edge2)) ) ) +
     new_frame_set( '-', frame_event );
END pulse_high;


FUNCTION pulse_high_skew( t0, t1, t2 : TIME ) RETURN frame_set IS

   CONSTANT edge0 : event_time := etime( t0 );
   CONSTANT edge1 : event_time := etime( t1 );
   CONSTANT edge2 : event_time := etime( t2 );

BEGIN
   ASSERT t0 < t1 AND t1 < t2
   REPORT "Timing violation in Pulse_High frames." &
          "The inequality: T0 < T1 < T2 Must hold."
   SEVERITY FAILURE;
   RETURN
     new_frame_set( 'X', frame_event ) +
     new_frame_set( '0', frame_event( ( drive_0, edge0)   ) ) +
     new_frame_set( '1', frame_elist( ((drive_0, edge0),
                                        (drive_1, edge1),
                                        (drive_0, edge2)) ) ) +
     new_frame_set( 'Z', frame_event ) +
     new_frame_set( 'W', frame_event ) +
        new_frame_set( 'L', frame_event( ( drive_L, edge0)   ) ) +
        new_frame_set( 'H', frame_elist( ((drive_L, edge0),
                                           (drive_H, edge1),
                                           (drive_L, edge2)) ) ) +
        new_frame_set( '-', frame_event );
    END pulse_high_skew;


    FUNCTION window( t1, t2 : TIME ) RETURN frame_set IS

      CONSTANT edge0 : event_time := etime( 0 ns );
      CONSTANT edge1 : event_time := etime( t1 );
      CONSTANT edge2 : event_time := etime( t2 );

    BEGIN
      ASSERT t1 < t2
      REPORT "Timing violation in Window frames." &
             "The inequality: T1 < T2 Must hold."
      SEVERITY FAILURE;
      RETURN
        new_frame_set( 'X', frame_elist( ((dont_care, edge0),
                                           (sense_X, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( '0', frame_elist( ((dont_care, edge0),
                                           (sense_0, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( '1', frame_elist( ((dont_care, edge0),
                                           (sense_1, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( 'Z', frame_elist( ((dont_care, edge0),
                                           (sense_Z, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( 'W', frame_elist( ((dont_care, edge0),
                                           (sense_W, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( 'L', frame_elist( ((dont_care, edge0),
                                           (sense_L, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( 'H', frame_elist( ((dont_care, edge0),
                                           (sense_H, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( '-', frame_event( ( dont_care, edge0)   ) );
    END window;

    FUNCTION window_skew( t0, t1, t2 : TIME ) RETURN frame_set IS

      CONSTANT edge0 : event_time := etime( t0 );
      CONSTANT edge1 : event_time := etime( t1 );
      CONSTANT edge2 : event_time := etime( t2 );

    BEGIN
      ASSERT t0 < t1 AND t1 < t2
      REPORT "Timing violation in Window frames." &
             "The inequality: T0 < T1 < T2 Must hold."
      SEVERITY FAILURE;
      RETURN
        new_frame_set( 'X', frame_elist( ((dont_care, edge0),
                                           (sense_X, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( '0', frame_elist( ((dont_care, edge0),
                                           (sense_0, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( '1', frame_elist( ((dont_care, edge0),
                                           (sense_1, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( 'Z', frame_elist( ((dont_care, edge0),
                                           (sense_Z, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( 'W', frame_elist( ((dont_care, edge0),
                                           (sense_W, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( 'L', frame_elist( ((dont_care, edge0),
                                           (sense_L, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( 'H', frame_elist( ((dont_care, edge0),
                                           (sense_H, edge1),
                                           (dont_care, edge2)) ) ) +
        new_frame_set( '-', frame_event( ( dont_care, edge0)   ) );
    END window_skew;

END waves_1164_frames;
```

APPENDIX 3: TESTBENCH UTILITIES

```
LIBRARY waves_std;
USE waves_std.waves_system.ALL;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

PACKAGE waves_1164_utilities IS

----------------------------------------
-- Procedure and Function Declarations --
----------------------------------------
--

  -- This function converts a waves port element to 1164 std_logic bit.
  -- The Translation is specific to the order of logic value definitions
  -- defined in the waves_1164_logic_values package
  --
  FUNCTION stim_1164( port_element : system_waves_port)
          RETURN STD_LOGIC;


  -- This function converts a waves port element to 1164 std_logic bits
  -- The Translation is specific to the order of logic value definitions
  -- defined in the waves_1164_logic_values package
  --
  FUNCTION stim_1164( port_list : system_waves_port_list)
          RETURN STD_ULOGIC_VECTOR;


  -- This function converts a waves port list to 1164 std_logic values
  -- The Translation is specific to the order of logic value definitions
  -- defined in the waves_1164_logic_values package
  --
  FUNCTION stim_1164( port_list : system_waves_port_list)
          RETURN STD_LOGIC_VECTOR;


  -- This function converts a waves port element to 1164 std_logic bits
  -- IT ALSO strips off the drive values and replaces them with '-'
  -- The Translation is specific to the order of logic value definitions
  -- defined in the waves_1164_logic_values package
  --
  FUNCTION expect_1164( port_element : system_waves_port)
          RETURN STD_ULOGIC;


  -- This function converts a waves port list to 1164 std_logic_vectors
  -- IT ALSO strips off the drive values and replaces them with '-'
  -- The Translation is specific to the order of logic value definitions
  -- defined in the waves_1164_logic_values package
  --
  FUNCTION expect_1164( port_list : system_waves_port_list)
          RETURN STD_ULOGIC_VECTOR;


  -- This function converts a waves port element to 1164 std_logic bits
  -- IT ALSO strips off the sense values and replaces them with 'Z'
  -- The Translation is specific to the order of logic value definitions
  -- defined in the waves_1164_logic_values package
  --
  FUNCTION bi_dir_1164( port_element : system_waves_port)
          RETURN STD_LOGIC;
```

```
   -- This function converts a waves port list to 1164 std_logic_vectors
   -- IT ALSO strips off the sense values and replaces them with 'Z'
   -- The Translation is specific to the order of logic value definitions
   -- defined in the waves_1164_logic_values package
   --
   FUNCTION bi_dir_1164( port_list : system_waves_port_list)
          RETURN STD_LOGIC_VECTOR;


   -- This function evaluates two 1164 std_logic bits for compatibility
   -- The actual data bit is evaluated to determine if it is compatible
   -- to a expected or predicted value.  The order must be preserved.
   -- example if actual data is '-' and expected data '1' result = false
   --          if actual data is '1' and expected data '-' result = true
   --
   FUNCTION compatible( actual: STD_LOGIC;
                      expected : STD_ULOGIC )
          RETURN BOOLEAN;


   -- This function evaluates two 1164 std_logic_vectors for compatibility
   -- The actual data bit is evaluated to determine if it is compatible
   -- to a expected or predicted value.  The order must be preserved.
   -- example if actual data is '-' and expected data '1' result = false
   --          if actual data is '1' and expected data '-' result = true
   --
   FUNCTION compatible( actual: STD_ULOGIC_VECTOR;
                      expected : STD_ULOGIC_VECTOR)
          RETURN BOOLEAN;


   -- This function evaluates two 1164 std_logic_vectors for compatibility
   -- The actual data bit is evaluated to determine if it is compatible
   -- to a expected or predicted value.  The order must be preserved.
   -- example if actual data is '-' and expected data '1' result = false
   --          if actual data is '1' and expected data '-' result = true
   --
   FUNCTION compatible( actual: STD_LOGIC_VECTOR;
                      expected : STD_ULOGIC_VECTOR)
          RETURN BOOLEAN;


END waves_1164_utilities;

PACKAGE BODY waves_1164_utilities IS

-------------------------------
-- Internal Type Definitions --
-------------------------------
--
  TYPE boolean_matrix IS ARRAY ( STD_ULOGIC, STD_ULOGIC ) OF BOOLEAN;
  -- Define Compatible Table.     (Actual, Expected)
  --
  CONSTANT compatible_table : boolean_matrix :=
  --   EXPECTED   EXPECTED   EXPECTED   EXPECTED   EXPECTED   EXPECTED
  -- U       X        0        1        Z        W        L        H        -
(( TRUE,  FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE ),  -- 'U'
 ( FALSE, TRUE,  FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE ),  -- 'X'  A
 ( FALSE, TRUE,  TRUE,  FALSE, FALSE, FALSE, FALSE, FALSE, TRUE ),  -- '0'  C
 ( FALSE, TRUE,  FALSE, TRUE,  FALSE, FALSE, FALSE, FALSE, TRUE ),  -- '1'  T
 ( FALSE, FALSE, FALSE, FALSE, TRUE,  FALSE, FALSE, FALSE, TRUE ),  -- 'Z'  U
 ( FALSE, FALSE, FALSE, FALSE, FALSE, TRUE,  FALSE, FALSE, TRUE ),  -- 'W'  A
 ( FALSE, FALSE, FALSE, FALSE, FALSE, TRUE,  TRUE,  FALSE, TRUE ),  -- 'L'  L
 ( FALSE, FALSE, FALSE, FALSE, FALSE, TRUE,  FALSE, TRUE,  TRUE ),  -- 'H'
 ( FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE )); -- '-'
--

  TYPE sim_code_array IS ARRAY (NATURAL RANGE <> ) OF STD_ULOGIC;

  CONSTANT translate : sim_code_array  :=('-','X','0','1','Z','W','L','H',
                                              'X','0','1','Z','W','L','H');
                                       --  0   1   2   3   4   5   6   7
                                       --      8   9  10  11  12  13  14
```

```
--
----------------------------
-- Procedure and Function --
----------------------------

  -- This function converts a waves port element to 1164 std_logic bits
  -- The Translation is specific to the order of logic value definitions
  -- defined in the waves_1164_logic_values package
  --
  FUNCTION stim_1164( port_element : system_waves_port)
          RETURN STD_LOGIC IS
  BEGIN
    RETURN translate( port_element.l_value );
  END stim_1164;


  -- This function converts a waves port list to 1164 std_logic_vectors
  -- The Translation is specific to the order of logic value definitions
  -- defined in the waves_1164_logic_values package
  --
  FUNCTION stim_1164( Port_list : system_waves_port_list)
          RETURN STD_ULOGIC_VECTOR IS

  VARIABLE r : STD_ULOGIC_VECTOR(port_list'RANGE);
  BEGIN
    FOR i IN port_list'RANGE LOOP
      r(i):= translate( port_list(i).l_value );
    END LOOP;
    RETURN r;
  END stim_1164;


  -- This function converts a waves port list to 1164 std_logic_vectors
  -- The Translation is specific to the order of logic value definitions
  -- defined in the waves_1164_logic_values package
  --
  FUNCTION stim_1164( port_list : system_waves_port_list)
          RETURN STD_LOGIC_VECTOR IS

  VARIABLE r : STD_LOGIC_VECTOR(port_list'RANGE);
  BEGIN
    FOR i IN port_list'RANGE LOOP
      r(i):= translate( port_list(i).l_value );
    END LOOP;
    RETURN r;
  END stim_1164;


  -- This function converts a waves port element to 1164 std_logic bits
  -- IT ALSO strips off the drive values and replaces them with '-'
  -- The Translation is specific to the order of logic value definitions
  -- defined in the waves_1164_logic_values package
  --
  FUNCTION expect_1164( port_element : system_waves_port)
          RETURN STD_ULOGIC IS

  VARIABLE result : STD_ULOGIC;
  BEGIN
    IF (port_element.l_value < 8 ) THEN
      result:= translate( port_element.l_value );
    ELSE
      result:='-';
    END IF;
    RETURN result;
  END expect_1164;
```

```
-- This function converts a waves port list to 1164 std_logic_vectors
-- IT ALSO strips off the drive values and replaces them with '-'
-- The Translation is specific to the order of logic value definitions
-- defined in the waves_1164_logic_values package
--
FUNCTION expect_1164( port_list : system_waves_port_list)
        RETURN STD_ULOGIC_VECTOR IS

VARIABLE r  : STD_ULOGIC_VECTOR(port_list'RANGE);
BEGIN
  FOR i IN port_list'RANGE LOOP
    IF (port_list( i ).l_value < 8 ) THEN
      r(i):= Translate( port_list(i).l_value );
    ELSE
      r(i):='-';
    END IF;
  END LOOP;
  RETURN r;
END expect_1164;


-- This function converts a waves port element to 1164 std_logic bits
-- IT ALSO strips off the sense values and replaces them with 'Z'
-- The Translation is specific to the order of logic value definitions
-- defined in the waves_1164_logic_values package
--
FUNCTION bi_dir_1164( port_element : system_waves_port)
        RETURN STD_LOGIC IS

VARIABLE result  : STD_LOGIC;
BEGIN
  IF (port_element.l_value > 7) THEN
    result:= translate( port_element.l_value );
  ELSE
    result:='Z';
  END IF;
  RETURN result;
END bi_dir_1164;


-- This function converts a waves port list to 1164 std_logic_vectors
-- IT ALSO strips off the sense values and replaces them with 'Z'
-- The Translation is specific to the order of logic value definitions
-- defined in the waves_1164_logic_values package
--
FUNCTION bi_dir_1164( port_list : system_waves_port_list)
        RETURN STD_LOGIC_VECTOR IS

VARIABLE r  : STD_LOGIC_VECTOR(port_list'RANGE);
BEGIN
  FOR i IN port_list'RANGE LOOP
    IF (Port_list( i ).l_value > 7 ) THEN
      r(i):= translate( port_list(i).l_value );
    ELSE
      r(i):='Z';
    END IF;
  END LOOP;
  RETURN r;
END bi_dir_1164;


-- This function evaluates two 1164 std_logic bits for compatibility
-- The actual data bit is evaluated to determine if it is compatible
-- to a expected or predicted value.  The order must be preserved.
--
FUNCTION compatible( actual  : STD_LOGIC;
                     expected  : STD_ULOGIC )
        RETURN BOOLEAN IS

BEGIN
  RETURN ( compatible_table(actual, expected) );
END compatible;
```

```
  -- This function evaluates two 1164 std_logic vectors for compatibility
  -- The actual data bit is evaluated to determine if it is compatible
  -- to a expected or predicted value.  The order must be preserved.
  --
  FUNCTION compatible(  actual  : STD_ULOGIC_VECTOR;
                      expected  : STD_ULOGIC_VECTOR)
         RETURN BOOLEAN IS

  ALIAS a :   STD_ULOGIC_VECTOR (actual'LENGTH-1 DOWNTO 0) IS actual;
  ALIAS e :   STD_ULOGIC_VECTOR (expected'LENGTH-1 DOWNTO 0) IS expected;

  BEGIN
    ASSERT e'LENGTH = a'LENGTH
       REPORT "Vector Length's Incompatable in Compatible Function"
       SEVERITY FAILURE;
    FOR index IN (expected'LENGTH -1) DOWNTO 0 LOOP
      IF NOT compatible_table( a(index), e(index)) THEN
        RETURN FALSE;
      END IF;
    END LOOP;
    RETURN TRUE;
  END compatible;


  -- This function evaluates two 1164 std_logic vectors for compatibility
  -- The actual data bit is evaluated to determine if it is compatible
  -- to a expected or predicted value.  The order must be preserved.
  --
  FUNCTION compatible(  actual  : STD_LOGIC_VECTOR;
                      expected  : STD_ULOGIC_VECTOR)
         RETURN BOOLEAN IS

  ALIAS a :   STD_LOGIC_VECTOR (actual'LENGTH-1 DOWNTO 0) IS actual;
  ALIAS e :   STD_ULOGIC_VECTOR (expected'LENGTH-1 DOWNTO 0) IS expected;

  BEGIN
    ASSERT e'LENGTH = a'LENGTH
       REPORT "Vector Length's Incompatable in Compatible Function"
       SEVERITY FAILURE;
    FOR index IN (expected'LENGTH -1) DOWNTO 0 LOOP
      IF NOT compatible_table( a(index), e(index)) THEN
        RETURN FALSE;
      END IF;
    END LOOP;
    RETURN TRUE;
  END compatible;

END waves_1164_utilities;
```

APPENDIX 4: TESTBENCH USER'S GUIDE

A4.1 VHDL Model

```
-- This is a behavioral model for a Positive edge triggered D Flip Flop
-- for a WAVES example.

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY d_flip_flop IS
  PORT ( clock   :IN  STD_LOGIC ;
         d       :IN  STD_LOGIC ;
         q       :OUT STD_LOGIC ;
         q_bar   :OUT STD_LOGIC);
END d_flip_flop ;

ARCHITECTURE behavioral OF d_flip_flop IS

BEGIN

main : PROCESS ( clock )

  BEGIN

    IF clock = '1' THEN
      q      <= d ;
      q_bar  <= NOT(d) ;
    ELSE
      NULL;
    END IF;

  END PROCESS;

END behavioral;
```

A4.2 WAVES HEADER FILE

```
-- **************************************************
--
-- ******** Header File for Entity: d_flip_flop
--
-- **************************************************
-- **************************************************
--
-- Data Set Identification Information
--
TITLE          A General Description
DEVICE_ID      d_flip_flop

DATE           Wed Sep  6 15:09:10 1995
ORIGIN         Company X Design Team
AUTHOR         Company or Person
AUTHOR         Maybe Multiple ... Companies or People
DATE           Wed Sep  6 15:09:10 1995
ORIGIN         Modified by Company X Design Team
AUTHOR         Who did it Company or Person

OTHER          Any general comments you want
OTHER          Built Using the WAVES-VHDL 1164 STD Libraries
--
-- Data Set Construction Information
--
waves_filename    ./waves_pins.vhd                          WORK
LIBRARY           waves_1164;
USE               waves_1164.waves_1164_pin_codes.ALL;
USE               waves_1164.waves_1164_logic_value.ALL;
USE               waves_1164.waves_interface.ALL;
USE               WORK.uut_test_pins.ALL;
waves_unit        waves_objects                             WORK
waves_filename    ./waves_wgen.vhd                          WORK
--
external_filename    d_flip_flop_vectors.txt                vectors
--
waveform_generator_procedure      WORK.waves_d_flip_flop.waveform
```

A4.3 WAVES PINS PACKAGE

```
-- ******** This File Was Automatically Generated   ********
-- ******** By The WAVES-VHDL Tool Set       ********
-- ******** Generated for Entity: d_flip_flop
-- ******** This File Was Generated on: Wed Sep  6 15:09:10 1995
--
--
PACKAGE uut_test_pins IS
TYPE test_pins IS (clock, d, q, q_bar);
END uut_test_pins;
```

A4.4 WAVES GENERATOR PACKAGE

```
-- ******** This File Was Automatically Generated  ********
-- ******** By The WAVES-VHDL Tool Set         ********
-- ******** Generated for Entity: d_flip_flop
-- ******** This File Was Generated on: Wed Sep  6 15:09:10 1995
--
--

LIBRARY waves_std;
USE waves_std.waves_standard.ALL;

USE STD.TEXTIO.ALL;
LIBRARY waves_1164;
USE waves_1164.waves_1164_frames.ALL;
USE waves_1164.waves_1164_pin_codes.ALL;
USE waves_1164.waves_interface.ALL;
USE WORK.waves_objects.ALL;
USE WORK.uut_test_pins.ALL;

PACKAGE wgp_d_flip_flop IS
     PROCEDURE  waveform(SIGNAL wpl : INOUT waves_port_list);
END wgp_d_flip_flop;

----------------------------------------------------

PACKAGE BODY wgp_d_flip_flop IS

-- This is the uut pin declaration pin and ordering
-- Remember you need to match the External file to This order
--
-- clock, D, Q, Q_bar

    PROCEDURE  waveform(SIGNAL wpl : INOUT waves_port_list) IS

       FILE vector_file  : TEXT IS IN "dff_vect.txt";

       VARIABLE vector  : file_slice := new_file_slice;

        -- declare time constants to use or use time literals
        -- constants or time literals can be used as the frame time values

         CONSTANT outputs: pinset:= new_pinset((q, q_bar));

         CONSTANT inputs: pinset:= new_pinset((d));


        CONSTANT vector_fsa : frame_set_array :=
            new_frame_set_array(pulse_high( 5 ns, 15 ns), clock) +
            new_frame_set_array(non_return( 0 ns), inputs) +
            new_frame_set_array(window( 10 ns, 20 ns), outputs);

       VARIABLE timing : time_data := new_time_data(vector_fsa);

      BEGIN
        LOOP
          read_file_slice (vector_file, vector);  -- get first vector
          EXIT WHEN vector.end_of_file;
          apply(wpl, vector.codes.ALL, delay(vector.fs_time), timing);
        END LOOP;

      END waveform;

END wgp_d_flip_flop;
```

A4.5 WAVES TESTBENCH CODE ·

```
-- ******** This File Was Automatically Generated  ********
-- ******** By The WAVES-VHDL Tool Set          ********
-- ******** Generated for Entity: d_flip_flop
-- ******** This File Was Generated on: Wed Sep  6 15:09:12 1995
--
--
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

LIBRARY waves_1164;
USE waves_1164.waves_1164_utilities.ALL;

USE WORK.uut_test_pins.ALL;
USE WORK.waves_objects.ALL;

USE WORK.wgp_d_flip_flop.ALL;
-- Include component libary references here
-- User Must Modify And ADD component libary references here
-- Include component libary references here

ENTITY test_bench IS
END test_bench;


ARCHITECTURE d_flip_flop_test OF test_bench IS


   --*******************************************************
   --***********CONFIGURATION SPECIFICATION ***************
   --*******************************************************

   COMPONENT d_flip_flop
    PORT ( clock                  :IN   STD_LOGIC;
           d                      :IN   STD_LOGIC;
           q                      :OUT  STD_LOGIC;
           q_bar                  :OUT  STD_LOGIC);
     END COMPONENT;

  -- Modify entity use statement
  -- User Must Modify modify and declare correct
  --   .. Architecture, Library, Component ..
  -- Modify entity use statement
FOR ALL:d_flip_flop USE ENTITY WORK.d_flip_flop(behavioral);

   --***********************************************************
   -- stimulus signals for the waveforms mapped into UUT INPUTS
   --***********************************************************

    SIGNAL wav_stim_clock        :STD_LOGIC;
    SIGNAL wav_stim_d            :STD_LOGIC;

   --*****************************************************
   -- Expected signals used in monitoring the UUT OUTPUTS
   --*****************************************************

    SIGNAL fail_signal           :STD_LOGIC;
    SIGNAL wav_expect_q          :STD_LOGIC;
    SIGNAL wav_expect_q_bar      :STD_LOGIC;

   --*******************************************************
   -- UUT Output signals used In Monitoring ACTUAL Values
   --*******************************************************

    SIGNAL actual_q              :STD_LOGIC;
    SIGNAL actual_q_bar          :STD_LOGIC;

   --**********************************************************
   -- Bi_directional signals used  for stimulus signals mapped
   -- into UUT INPUTS and also monitoring the UUT OUTPUTS
   --**********************************************************
```

```
--
--
-- No Bidirectional Pins On UUT

   --*****************************************************************
   -- WAVES signals OUTPUTing each slice of the waves port list
   --*****************************************************************

        SIGNAL wpl   : waves_port_list;

BEGIN
   --
   --*****************************************************************
   -- process that generates the WAVES waveform
   --*****************************************************************

       waves: waveform(wpl);

   --*****************************************************************
   -- processes that convert the WPL values to 1164 Logic Values
   --*****************************************************************

  wav_stim_clock              <= stim_1164(wpl.wpl( 1 ));
  wav_stim_d                  <= stim_1164(wpl.wpl( 2 ));
  wav_expect_q                <= expect_1164(wpl.wpl( 3 ));
  wav_expect_q_bar            <= expect_1164(wpl.wpl( 4 ));

   --******************************************
   -- UUT Port Map - Name Symantics Denote Usage
   --******************************************

  u1: d_flip_flop
   PORT MAP(
    clock                     => wav_stim_clock,
    d                         => wav_stim_d,
    q                         => actual_q,
    q_bar                     => actual_q_bar);


   --*************************************************************
   -- Monitor Processes To Verify The UUT Operational Response
   --*************************************************************

 monitor_q:
   PROCESS(actual_q, wav_expect_q)
   BEGIN
        ASSERT(compatible (actual => actual_q,
                          expected => wav_expect_q))
         REPORT "Error on Q output" SEVERITY WARNING;

   IF ( compatible ( actual_q,     wav_expect_q) ) THEN
     fail_signal <='L'; ELSE fail_signal <='1';
   END IF;
   END PROCESS;


 monitor_q_bar:
   PROCESS(actual_q_bar, wav_expect_q_bar)
   BEGIN
        ASSERT(compatible (actual => actual_q_bar,
                          expected => wav_expect_q_bar))
         REPORT "Error on Q_bar output" SEVERITY WARNING;

   IF ( compatible ( actual_q_bar,     wav_expect_q_bar) ) THEN
     fail_signal <='L'; ELSE fail_signal <='1';
   END IF;
   END PROCESS;

END d_flip_flop_test;
```

A4.6 External Vector File

```
%clock data  Q   QBAR
    1    1   1    0 : 20 ns;
    0    0   1    0 : 20 ns;
    1    0   0    1 : 20 ns;
    1    1   1    0 : 20 ns;
    1    0   0    1 : 20 ns;
```

APPENDIX 5: 54/74180 8 - BIT PARITY GENERATOR/CHECKER

A5.1 VHDL Model

```
-- TITLE: Fairchild 54/74180 8-BIT PARITY GENERATOR/CHECKER
-- DATE : 5 May 1995
--
-- VERSION : 2.0
-- FILENAME: parity_gen.vhd
-- FUNCTION: Entity, Architecture for the Fairchild 54/74180
--            8-BIT PARITY GENERATOR/CHECKER - Behavioral Model.
--
-- AUTHOR       : Steven Drager
-- ORGANIZATION: Rome Laboratory
--
-- PURPOSE AND USE: This was written as an example to help build the Rome
--                  Laboratory WAVES tools.
--
-- TIMING: None
-- NOTES :
--
--
-- DEVELOPMENT PLATFORM : Pentium w/WIN'95
-- VHDL SOFTWARE VERSION: MTI VSYSTEM/WINDOWS v4.2f
--
-- HISTORY:
--16 May 95 - v2.0 - Includes timing
-- 9 May 95 - v1.5 - Changed to case statement, still no timing.
-- 5 May 95 - v1.0 - Initial version, functional only, no timing.
--
--

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY parity_generator_checker IS
PORT(   data_input              :IN     STD_LOGIC_VECTOR(0 TO 7);
        odd_input               :IN     STD_LOGIC;
        even_input              :IN     STD_LOGIC;
        odd_parity_output       :OUT    STD_LOGIC;
        even_parity_output      :OUT    STD_LOGIC);
END parity_generator_checker;

ARCHITECTURE behavioral OF parity_generator_checker IS

BEGIN

PROCESS(data_input, odd_input, even_input)

VARIABLE parity                 :STD_LOGIC;
VARIABLE controls               :STD_LOGIC_VECTOR(0 TO 2);
VARIABLE tplh_odd               :TIME;
VARIABLE tphl_odd               :TIME;
VARIABLE tplh_even              :TIME;
VARIABLE tphl_even              :TIME;
```

```
BEGIN

-- Generate Timing

IF odd_input = '0' AND data_input'EVENT THEN
       tplh_odd   := 48 ns;
       tphl_odd   := 38 ns;
       tplh_even  := 60 ns;
       tphl_even  := 68 ns;
END IF;
IF even_input = '0' AND data_input'EVENT THEN
       tplh_odd   := 60 ns;
       tphl_odd   := 68 ns;
       tplh_even  := 48 ns;
       tphl_even  := 38 ns;
END IF;
IF NOT data_input'EVENT AND (odd_input'EVENT OR even_input'EVENT) THEN
       tplh_odd   := 20 ns;
       tphl_odd   := 10 ns;
       tplh_even  := 20 ns;
       tphl_even  := 10 ns;
END IF;

-- Calculate Parity

parity := '0';

FOR x IN data_input'RANGE LOOP
  parity := parity XOR data_input(x);
END LOOP;


-- Assign Output

controls := parity & odd_input & even_input;

CASE controls IS
WHEN "100" | "000" =>
       odd_parity_output  <= '1' AFTER tplh_odd;
       even_parity_output <= '1' AFTER tplh_even;
WHEN "101" | "010" =>
       odd_parity_output  <= '1' AFTER tplh_odd;
       even_parity_output <= '0' AFTER tphl_even;
WHEN "110" | "001" =>
       odd_parity_output  <= '0' AFTER tphl_odd;
       even_parity_output <= '1' AFTER tplh_even;
WHEN "111" | "011" =>
       odd_parity_output  <= '0' AFTER tphl_odd;
       even_parity_output <= '0' AFTER tphl_even;
WHEN OTHERS =>
        NULL;
END CASE;

END PROCESS;
END behavioral;
```

A5.2 WAVES HEADER FILE

```
-- ************************************************
--
-- ******** Header File for Entity: parity_generator_checker
--
-- ************************************************
-- ************************************************
--
-- Data Set Identification Information
--
TITLE          A General Description
DEVICE_ID      parity_generator_checker

DATE           Thu Sep  7 09:36:56 1995
ORIGIN         Company X Design Team
AUTHOR         Company or Person
AUTHOR         Maybe Multiple ... Companies or People
DATE           Thu Sep  7 09:36:56 1995
ORIGIN         Modified by Company X Design Team
AUTHOR         Who did it Company or Person

OTHER          Any general comments you want
OTHER          Built Using the WAVES-VHDL 1164 STD Libraries
--
-- Data Set Construction Information
--
waves_filename    ./waves_pins.vhd                        WORK
LIBRARY           waves_1164;
USE               waves_1164.waves_1164_pin_codes.ALL;
USE               waves_1164.waves_1164_logic_value.ALL;
USE               waves_1164.waves_interface.ALL;
USE               WORK.uut_test_pins.ALL;
waves_unit        waves_objects                           WORK
waves_filename    ./waves_wgen.vhd                        WORK
--
external_filename   parity_generator_checker_vectors.txt  vectors
--
waveform_generator_procedure WORK.waves_parity_generator_checker.waveform
```

A5.3 WAVES PINS PACKAGE

```
-- ******** This File Was Automatically Generated  ********
-- ******** By The WAVES-VHDL Tool Set       ********
-- ******** Generated for Entity: parity_generator_checker
-- ******** This File Was Generated on: Thu Sep  7 09:36:56 1995
--
--
PACKAGE uut_test_pins IS
TYPE test_pins IS (data_input_0, data_input_1, data_input_2, data_input_3,
          data_input_4, data_input_5, data_input_6, data_input_7, odd_input,
          even_input, odd_parity_output, even_parity_output);
END uut_test_pins;
```

A5.4 WAVES Generator Package

```
-- ******** This File Was Automatically Generated  ********
-- ******** By The WAVES-VHDL Tool Set       ********
-- ******** Generated for Entity: parity_generator_checker
-- ******** This File Was Generated on: Thu Sep  7 09:36:57 1995
--
--

LIBRARY waves_std;
USE waves_std.waves_standard.ALL;

USE STD.TEXTIO.ALL;
LIBRARY waves_1164;
USE waves_1164.waves_1164_frames.ALL;
USE waves_1164.waves_1164_pin_codes.ALL;
USE waves_1164.waves_interface.ALL;
USE WORK.waves_objects.ALL;
USE WORK.uut_test_pins.ALL;

PACKAGE wgp_parity_generator_checker IS
     PROCEDURE  waveform(SIGNAL wpl : INOUT waves_port_list);
END wgp_parity_generator_checker;

--------------------------------------------------------

PACKAGE BODY wgp_parity_generator_checker IS

-- This is the uut pin declaration pin and ordering
-- Remember you need to match the External file to This order
--
-- Data_input_0, Data_input_1, Data_input_2, Data_input_3, Data_input_4,
-- Data_input_5, Data_input_6, Data_input_7, Odd_input, Even_input,
-- Odd_parity_output, Even_parity_output

    PROCEDURE  waveform(SIGNAL wpl : INOUT waves_port_list) IS

       FILE vector_file : TEXT IS IN "vectors.txt";

       VARIABLE vector : file_slice := new_file_slice;

        -- declare time constants to use or use time literals
        -- constants or time literals can be used as the frame time values

         CONSTANT data_input: pinset:= new_pinset(( data_input_0, data_input_1,
                      data_input_2, data_input_3, data_input_4, data_input_5,
                      data_input_6, data_input_7));

         CONSTANT outputs: pinset:= new_pinset((odd_parity_output,
                               even_parity_output));

         CONSTANT in_pins: pinset:= new_pinset((odd_input,
                               even_input));

         CONSTANT inputs: pinset:= in_pins OR data_input;

        CONSTANT vector_fsa : frame_set_array :=
            new_frame_set_array(non_return( 0 ns), inputs) +
            new_frame_set_array(window( 70 ns, 100 ns), outputs);

       VARIABLE timing : time_data := new_time_data(vector_fsa);

      BEGIN
        LOOP
          read_file_slice (vector_file, vector);  -- get first vector
          EXIT WHEN vector.end_of_file;
          apply(wpl, vector.codes.ALL, delay(vector.fs_time), timing);
    --     or use internal slice time format as below
    --     apply(wpl, vector.codes.all, Delay( ns), timing);
        END LOOP;

      END waveform;

END wgp_parity_generator_checker;
```

A5.5 WAVES TESTBENCH CODE

```
-- ******** This File Was Automatically Generated  ********
-- ******** By The WAVES-VHDL Tool Set        ********
-- ******** Generated for Entity: parity_generator_checker
-- ******** This File Was Generated on: Thu Sep  7 09:37:00 1995
--
--
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

LIBRARY waves_1164;
USE waves_1164.waves_1164_utilities.ALL;

USE WORK.uut_test_pins.ALL;
USE WORK.waves_objects.ALL;

USE WORK.wgp_parity_generator_checker.ALL;
-- Include component libary references here
-- User Must Modify And ADD component libary references here
-- Include component libary references here

ENTITY test_bench IS
END test_bench;


ARCHITECTURE parity_generator_checker_test OF test_bench IS

   --*****************************************************
   --***********CONFIGURATION SPECIFICATION ***************
   --*****************************************************

   COMPONENT parity_generator_checker
    PORT ( data_input               :IN   STD_LOGIC_VECTOR(  0 TO  7 );
           odd_input                :IN   STD_LOGIC;
           even_input               :IN   STD_LOGIC;
           odd_parity_output        :OUT  STD_LOGIC;
           even_parity_output       :OUT  STD_LOGIC);
     END COMPONENT;

  -- Modify entity use statement
  -- User Must Modify modify and declare correct
  --   .. Architecture, Library, Component ..
  -- Modify entity use statement
FOR ALL:parity_generator_checker USE ENTITY
       WORK.parity_generator_checker(behavioral);

   --*********************************************************
   -- stimulus signals for the waveforms mapped into UUT INPUTS
   --*********************************************************

    SIGNAL wav_stim_data_input          :STD_LOGIC_VECTOR(  0 TO  7 );
    SIGNAL wav_stim_odd_input           :STD_LOGIC;
    SIGNAL wav_stim_even_input          :STD_LOGIC;

   --****************************************************
   -- Expected signals used in monitoring the UUT OUTPUTS
   --****************************************************

    SIGNAL fail_signal                  :STD_LOGIC;
    SIGNAL wav_expect_odd_parity_output  :STD_LOGIC;
    SIGNAL wav_expect_even_parity_output :STD_LOGIC;

   --*****************************************************
   -- UUT Output signals used In Monitoring ACTUAL Values
   --*****************************************************

    SIGNAL actual_odd_parity_output      :STD_LOGIC;
    SIGNAL actual_even_parity_output     :STD_LOGIC;

   --*********************************************************
   -- Bi_directional signals used  for stimulus signals mapped
   -- into UUT INPUTS and also monitoring the UUT OUTPUTS
   --*********************************************************
```

```
--
--
-- No Bidirectional Pins On UUT

  --**************************************************************
  -- WAVES signals OUTPUTing each slice of the waves port list
  --**************************************************************

       SIGNAL wpl   : waves_port_list;

BEGIN
  --
  --**************************************************************
  -- process that generates the WAVES waveform
  --**************************************************************

      waves: waveform(wpl);

  --**************************************************************
  -- processes that convert the WPL values to 1164 Logic Values
  --**************************************************************

  wav_stim_data_input                 <= stim_1164(wpl.wpl( 1 TO 8 ));
  wav_stim_odd_input                  <= stim_1164(wpl.wpl( 9 ));
  wav_stim_even_input                 <= stim_1164(wpl.wpl( 10 ));
  wav_expect_odd_parity_output        <= expect_1164(wpl.wpl( 11 ));
  wav_expect_even_parity_output       <= expect_1164(wpl.wpl( 12 ));

  --****************************************
  -- UUT Port Map - Name Symantics Denote Usage
  --****************************************

  u1: parity_generator_checker
  PORT MAP(
    data_input                        => wav_stim_data_input,
    odd_input                         => wav_stim_odd_input,
    even_input                        => wav_stim_even_input,
    odd_parity_output                 => actual_odd_parity_output,
    even_parity_output                => actual_even_parity_output);

  --*******************************************************
  -- Monitor Processes To Verify The UUT Operational Response
  --*******************************************************

 monitor_odd_parity_output:
   PROCESS(actual_odd_parity_output, wav_expect_odd_parity_output)
   BEGIN
        ASSERT(compatible (actual => actual_odd_parity_output,
                          expected => wav_expect_odd_parity_output))
         REPORT "Error on Odd_parity_output output" SEVERITY WARNING;

   IF ( compatible ( actual_odd_parity_output,    wav_expect_odd_parity_output) ) THEN
     fail_signal <='L'; ELSE fail_signal <='1';
   END IF;
   END PROCESS;


 monitor_even_parity_output:
   PROCESS(actual_even_parity_output, wav_expect_even_parity_output)
   BEGIN
        ASSERT(compatible (actual => actual_even_parity_output,
                          expected => wav_expect_even_parity_output))
         REPORT "Error on Even_parity_output output" SEVERITY WARNING;

   IF ( compatible ( actual_even_parity_output,    wav_expect_even_parity_output) )
THEN
     fail_signal <='L'; ELSE fail_signal <='1';
   END IF;
   END PROCESS;

END parity_generator_checker_test;
```

A5.6 External Vector File

```
%DDDD DDDD     e
%aaaa aaaa  e ov
%tttt tttt ov de
%aaaa aaaa de dn
%|||| |||| dn ||
%iiii iiii || oo
%nnnn nnnn ii uu
%pppp pppp nn tt
%uuuu uuuu pp pp
%tttt tttt uu uu
%0123 4567 tt tt

 0000 0000 01 01 : 100 NS;
 0000 0001 01 10 : 100 NS;
 0000 0010 01 10 : 100 NS;
 0000 0011 01 01 : 100 NS;
 0000 0000 10 10 : 100 NS;
 0000 0001 10 01 : 100 NS;
 0000 0010 10 01 : 100 NS;
 0000 0011 10 10 : 100 NS;
 0000 0000 00 11 : 100 NS;
 0000 0001 00 11 : 100 NS;
 0000 0010 00 11 : 100 NS;
 0000 0011 00 11 : 100 NS;
 0000 0000 11 00 : 100 NS;
 0000 0001 11 00 : 100 NS;
 0000 0010 11 00 : 100 NS;
 0000 0011 11 00 : 100 NS;
```

APPENDIX 6: SN54/74ALS8169: SYNCHRONOUS 8 - BIT UP/DOWN BINARY COUNTER

A6.1 VHDL MODEL

```
-- TITLE: Texas Instruments SN54ALS8169 Synchronous 8-Bit Up/Down Binary Counter
-- DATE : 3 May 1995
--
-- VERSION : 2.1
-- FILENAME: sync_8_UDC_e_a.vhd
-- FUNCTION: Entity, Architecture for the Texas Instruments SN54ALS8169
--            Synchronous 8-Bit Up/Down Binary Counter - Behavioral Model.
--
-- AUTHOR       : Christopher Flynn
-- ORGANIZATION: Rome Laboratory
--
-- PURPOSE AND USE: This was written as an example to help build the Rome
--                   Laboratory WAVES tools.
--
-- TIMING: Max
-- NOTES : To implement the SN74ALS8169 simply modify the generic timing
--         statements.
--
-- DEVELOPMENT PLATFORM : Sun Sparc Station IPX
-- VHDL SOFTWARE VERSION: Cadence Leapfrog, v2.0
--
-- HISTORY:
-- 18 Apr 95 - v1.0 - Initial version, functional only, no timing.
--  3 May 95 - v2.0 - Final version with worst case timing.
-- 26 May 95 - v2.1 - Final version with worst case timing and To_Int and
--                    To_StdLogicVector functions.
--

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY eight_bit_sync_ud_cntr IS

  GENERIC ( data_in_setup    :TIME := 15 ns;
            enp_bar_setup    :TIME := 17 ns;
            ent_bar_setup    :TIME := 17 ns;
            load_bar_setup   :TIME := 15 ns;
            u_d_bar_setup    :TIME := 17 ns;
            tplh_q           :TIME := 18 ns;
            tplh_rco_bar     :TIME := 14 ns; -- For both CLK and ENT_bar
            tplh_rco_bar_ud  :TIME := 17 ns;
            tphl_rco_bar_ud  :TIME := 18 ns
          );

  PORT ( clk                :IN  STD_LOGIC ;
         load_bar           :IN  STD_LOGIC ;
         up_down_bar        :IN  STD_LOGIC ;
         ent_bar            :IN  STD_LOGIC ;
         enp_bar            :IN  STD_LOGIC ;
         hgfedcba           :IN  STD_LOGIC_VECTOR(7 DOWNTO 0) ;
         q                  :OUT STD_LOGIC_VECTOR(7 DOWNTO 0) ;
         rco_bar            :OUT STD_LOGIC
       );

END eight_bit_sync_ud_cntr;

ARCHITECTURE behavioral OF eight_bit_sync_ud_cntr IS

SIGNAL q_internal : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
```

```
-- Model Specific Functions: To_StdLogicVector
--                           To_Int

-- The To_StdLogicVector function converts an integer to an
-- 1164 Standard Logic Vector of desired length.

    FUNCTION to_stdlogicvector (oper: INTEGER; length: NATURAL)
           RETURN STD_LOGIC_VECTOR IS
        VARIABLE temp : INTEGER := oper;
        VARIABLE temp_result : STD_LOGIC_VECTOR (length -1 DOWNTO 0) :=
                                 (OTHERS => '0');
    BEGIN
        IF (oper = INTEGER'LEFT) THEN  --  Max negative integer
           temp_result(0) := '1';
           IF (length > 31) THEN
               temp_result(31) := '1';
           END IF;
           RETURN temp_result;
        END IF;

        FOR i IN 0 TO length - 1 LOOP
            IF (temp MOD 2) = 1 THEN
                temp_result(i) := '1';
            ELSE
                temp_result(i) := '0';
            END IF;
            EXIT WHEN temp = 0;
            IF temp > 0 THEN
                temp := temp / 2;
            ELSE
               temp := (temp - 1) / 2;
            END IF;
        END LOOP;
        RETURN temp_result;
    END to_stdlogicvector;

  -- The To_Int function converts an arbitrary length IEEE STD 1164
  -- std_logic_vector into an integer.  The format of the std_logic_vector
  -- is assumed to be two's complement.  This function assumes that VECTOR
  -- is a decending range, zero bounded vector (ie. VECTOR'length-1 downto 0)
  -- and that VECTOR'left is the MSB.

  FUNCTION to_int( vector : IN STD_LOGIC_VECTOR ) RETURN INTEGER IS

  VARIABLE result :INTEGER := 0;
  VARIABLE is_neg :BOOLEAN;
  VARIABLE temp   :STD_LOGIC_VECTOR( vector'RANGE ) := vector;

  BEGIN
      IF temp( temp'LEFT ) = '1' THEN
          is_neg := TRUE;
          temp := NOT temp;
      END IF;
      FOR i IN temp'RANGE LOOP
          IF temp( i ) = '1' THEN
              result := result + ( 2 ** i );
          END IF;
      END LOOP;
      IF is_neg THEN
          result := -( result );
          result :=  result - 1;
      END IF;
      RETURN result;
  END to_int;

BEGIN            -- Architecture behavioral.

clk_pulse : PROCESS ( clk, ent_bar, up_down_bar )

  VARIABLE preset       :STD_LOGIC_VECTOR(7 DOWNTO 0) ;
  VARIABLE internal_rco :STD_LOGIC ;

  BEGIN

  IF ( RISING_EDGE( clk ) AND NOT( ent_bar'EVENT ) AND
       NOT( up_down_bar'EVENT )) THEN

      -- Load Preset and Internal_RCO values.
      IF ( load_bar = '0' AND load_bar'STABLE( load_bar_setup) ) THEN

        IF hgfedcba'STABLE( data_in_setup ) THEN
```

```
          preset := hgfedcba ;
        ELSE
           ASSERT FALSE REPORT "Input data not stable !" SEVERITY WARNING;
          preset := "XXXXXXXX" ;
        END IF;

        IF (( preset = "11111111" AND up_down_bar = '1' ) OR
            ( preset = "00000000" AND up_down_bar = '0' )) THEN
          internal_rco := '0';
        ELSE
          internal_rco := '1';
        END IF;

      ELSIF ( ent_bar = '0' AND enp_bar = '0') THEN
        IF ( ent_bar'STABLE( ent_bar_setup ) AND
             enp_bar'STABLE( ent_bar_setup )) THEN
             -- Enable counting.

        IF up_down_bar'STABLE( u_d_bar_setup ) THEN
          IF up_down_bar = '1' THEN                      -- Count Up.
            preset := to_stdlogicvector((to_int(preset) + 1), preset'LENGTH);
          ELSIF up_down_bar = '0' THEN                   -- Count Down.
            preset := to_stdlogicvector((to_int(preset) - 1), preset'LENGTH);
          END IF;
        ELSE
          ASSERT FALSE REPORT "Up_Down_bar input not stable, unable to count !"
                 SEVERITY WARNING;
        END IF;

        IF (( preset = "11111111" AND q_internal = "11111110" ) OR
            ( preset = "00000000" AND q_internal = "00000001" )) THEN
          internal_rco := '0';
        ELSE
          internal_rco := '1';
        END IF;

        ELSE
          ASSERT FALSE REPORT "ENP_bar or ENT_bar not stable !" SEVERITY WARNING;
        END IF;

      END IF;

    q_internal <= preset ;
    q          <= preset AFTER tplh_q ;
    rco_bar    <= internal_rco AFTER tplh_rco_bar ;

    END IF;   -- IF rising_edge( CLK )

    IF ( RISING_EDGE( ent_bar ) AND NOT( clk'EVENT ) AND
         NOT( up_down_bar'EVENT )) THEN
           rco_bar <= '1' AFTER tplh_rco_bar ;
    ELSE
           NULL;
    END IF;

    IF ( RISING_EDGE( up_down_bar ) AND NOT( clk'EVENT ) AND
         NOT( ent_bar'EVENT )) THEN
      IF q_internal = "11111111" THEN
       rco_bar <= '0' AFTER tplh_rco_bar_ud ;
      ELSE
       rco_bar <= '1' AFTER tplh_rco_bar_ud ;
      END IF;
    ELSE
     NULL;
    END IF;

    IF ( FALLING_EDGE( up_down_bar ) AND NOT( clk'EVENT ) AND
         NOT( ent_bar'EVENT )) THEN
      IF q_internal = "00000000" THEN
        rco_bar <= '0' AFTER tphl_rco_bar_ud ;
      ELSE
        rco_bar <= '1' AFTER tphl_rco_bar_ud ;
      END IF;
    ELSE
     NULL;
    END IF;

  END PROCESS ; -- clk_pulse

END behavioral;
```

A6.2 WAVES HEADER FILE

```
-- ***************************************************
--
-- ******** Header File for Entity: eight_bit_sync_ud_cntr
--
-- ***************************************************
-- ***************************************************
--
-- Data Set Identification Information
--
TITLE         A General Description
DEVICE_ID     eight_bit_sync_ud_cntr

DATE          Wed Sep  6 14:41:27 1995
ORIGIN        Company X Design Team
AUTHOR        Company or Person
AUTHOR        Maybe Multiple ... Companies or People
DATE          Wed Sep  6 14:41:27 1995
ORIGIN        Modified by Company X Design Team
AUTHOR        Who did it Company or Person

OTHER         Any general comments you want
OTHER         Built Using the WAVES-VHDL 1164 STD Libraries
--
-- Data Set Construction Information
--
waves_filename    waves_pins.vhd                          WORK
LIBRARY           waves_1164;
USE               waves_1164.waves_1164_pin_codes.ALL;
USE               waves_1164.waves_1164_logic_value.ALL;
USE               waves_1164.waves_interface.ALL;
USE               WORK.uut_test_pins.ALL;
waves_unit        waves_objects                             WORK
waves_filename    waves_wgen.vhd                          WORK
--
external_filename   synctr_vectors.txt    vectors
--
waveform_generator_procedure      WORK.waves_eight_bit_sync_ud_cntr.waveform
```

A6.3 WAVES PINS PACKAGE

```
-- ******** This File Was Automatically Generated  ********
-- ******** By The WAVES-VHDL Tool Set      ********
-- ******** Generated for Entity: eight_bit_sync_ud_cntr
-- ******** This File Was Generated on: Wed Sep  6 14:41:27 1995
--
--
PACKAGE uut_test_pins IS
TYPE test_pins IS (clk, load_bar, up_down_bar, ent_bar, enp_bar,
          hgfedcba_7, hgfedcba_6, hgfedcba_5, hgfedcba_4, hgfedcba_3,
          hgfedcba_2, hgfedcba_1, hgfedcba_0, q_7, q_6, q_5, q_4, q_3, q_2,
          q_1, q_0, rco_bar);
END uut_test_pins;
```

A6.4 WAVES GENERATOR PACKAGE

```
-- ******** This File Was Automatically Generated   ********
-- ******** By The WAVES-VHDL TestBench Tool         ********
-- ******** Generated for Entity: eight_bit_sync_ud_cntr
-- ******** This File Was Generated on: Tue May  2 07:00:52 1995
--
--

LIBRARY waves_std;
USE waves_std.waves_standard.ALL;
LIBARY waves_1164;
USE STD.TEXTIO.ALL;
USE waves_1164.waves_1164_frames.ALL;
USE waves_1164.waves_1164_pin_codes.ALL;
USE waves_1164.waves_interface.ALL;
USE WORK.waves_objects.ALL;
USE WORK.uut_test_pins.ALL;

PACKAGE wgp_eight_bit_sync_ud_cntr IS
     PROCEDURE  waveform(SIGNAL wpl : INOUT waves_port_list);
END wgp_eight_bit_sync_ud_cntr;

-------------------------------------------------------

PACKAGE BODY wgp_eight_bit_sync_ud_cntr IS

-- This is the uut pin declaration pin and ordering
-- Remember you need to match the Eternal file to This order
--
-- CLK, LOAD_bar, Up_Down_bar, ENT_bar, ENP_bar, HGFEDCBA_7, HGFEDCBA_6,
-- HGFEDCBA_5, HGFEDCBA_4, HGFEDCBA_3, HGFEDCBA_2, HGFEDCBA_1, HGFEDCBA_0,
-- Q_7, Q_6, Q_5, Q_4, Q_3, Q_2, Q_1, Q_0, RCO_bar

    PROCEDURE  waveform(SIGNAL wpl : INOUT waves_port_list) IS

       FILE vector_file  : TEXT IS IN "synctr_vectors.txt";

       VARIABLE vector : file_slice := new_file_slice;

        -- declare time constants to use or use time literals
        -- constants or time literals can be used as the frame time values

         CONSTANT hgfedcba: pinset:= new_pinset(( hgfedcba_7, hgfedcba_6,
                    hgfedcba_5, hgfedcba_4, hgfedcba_3, hgfedcba_2, hgfedcba_1,
                    hgfedcba_0));

         CONSTANT q: pinset:= new_pinset(( q_7, q_6, q_5, q_4, q_3, q_2,
                     q_1, q_0));

         CONSTANT out_pins: pinset:= new_pinset((rco_bar));

         CONSTANT in_pins: pinset:= new_pinset((load_bar, up_down_bar));

         CONSTANT enable_pins : pinset := new_pinset((ent_bar, enp_bar));

         CONSTANT inputs: pinset:= in_pins OR hgfedcba;

         CONSTANT outputs: pinset:= out_pins OR q;


        CONSTANT vector_fsa : frame_set_array :=
            new_frame_set_array(pulse_high(15 ns, 30 ns), clk) +
            new_frame_set_array(non_return(20 ns), enable_pins) +
            new_frame_set_array(non_return(0 ns), inputs) +
            new_frame_set_array(window(20 ns, 27 ns), outputs);

       VARIABLE timing : time_data := new_time_data(vector_fsa);
       BEGIN
         LOOP
           read_file_slice (vector_file, vector);  -- get first vector
           EXIT WHEN vector.end_of_file;
           apply(wpl, vector.codes.ALL, delay(vector.fs_time), timing);
--          or use internal slice time format as below
--           apply(wpl, vector.codes.all, Delay( ns), timing);
         END LOOP;

       END waveform;

END wgp_eight_bit_sync_ud_cntr;
```

A6.5 WAVES TESTBENCH CODE

```
-- ******** This File Was Automatically Generated  ********
-- ******** By The WAVES-VHDL Tool Set       ********
-- ******** Generated for Entity: eight_bit_sync_ud_cntr
-- ******** This File Was Generated on: Wed Sep  6 14:41:32 1995
--
--
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

LIBRARY waves_1164;
USE waves_1164.waves_1164_utilities.ALL;

USE WORK.uut_test_pins.ALL;
USE WORK.waves_objects.ALL;

USE WORK.wgp_eight_bit_sync_ud_cntr.ALL;
-- Include component libary references here
-- User Must Modify And ADD component libary references here
-- Include component libary references here

ENTITY test_bench IS
END test_bench;


ARCHITECTURE eight_bit_sync_ud_cntr_test OF test_bench IS

    --*****************************************************
    --***********CONFIGURATION SPECIFICATION ***************
    --*****************************************************

    COMPONENT eight_bit_sync_ud_cntr
     PORT ( clk                              :IN   STD_LOGIC;
            load_bar                         :IN   STD_LOGIC;
            up_down_bar                      :IN   STD_LOGIC;
            ent_bar                          :IN   STD_LOGIC;
            enp_bar                          :IN   STD_LOGIC;
            hgfedcba                         :IN   STD_LOGIC_VECTOR(  7 DOWNTO  0 );
            q                                :OUT  STD_LOGIC_VECTOR(  7 DOWNTO  0 );
            rco_bar                          :OUT  STD_LOGIC);
      END COMPONENT;

 -- Modify entity use statement
 -- User Must Modify modify and declare correct
 --   .. Architecture, Library, Component ..
 -- Modify entity use statement
FOR ALL:eight_bit_sync_ud_cntr USE ENTITY WORK.eight_bit_sync_ud_cntr(behavioral);

   --*******************************************************
   -- stimulus signals for the waveforms mapped into UUT INPUTS
   --*******************************************************

    SIGNAL wav_stim_clk                      :STD_LOGIC;
    SIGNAL wav_stim_load_bar                 :STD_LOGIC;
    SIGNAL wav_stim_up_down_bar              :STD_LOGIC;
    SIGNAL wav_stim_ent_bar                  :STD_LOGIC;
    SIGNAL wav_stim_enp_bar                  :STD_LOGIC;
    SIGNAL wav_stim_hgfedcba                 :STD_LOGIC_VECTOR(  7 DOWNTO  0 );

   --****************************************************
   -- Expected signals used in monitoring the UUT OUTPUTS
   --****************************************************

    SIGNAL fail_signal                       :STD_LOGIC;
    SIGNAL wav_expect_q                      :STD_ULOGIC_VECTOR(  7 DOWNTO  0 );
    SIGNAL wav_expect_rco_bar                :STD_LOGIC;
```

```
  --*****************************************************
  -- UUT Output signals used In Monitoring ACTUAL Values
  --*****************************************************

    SIGNAL actual_q                         :STD_LOGIC_VECTOR(  7 DOWNTO  0 );
    SIGNAL actual_rco_bar                   :STD_LOGIC;

  --********************************************************
  -- Bi_directional signals used  for stimulus signals mapped
  -- into UUT INPUTS and also monitoring the UUT OUTPUTS
  --********************************************************

--
--
-- No Bidirectional Pins On UUT


  --**************************************************************
  -- WAVES signals OUTPUTing each slice of the waves port list
  --**************************************************************

        SIGNAL wpl   : waves_port_list;

BEGIN
  --
  --**************************************************************
  -- process that generates the WAVES waveform
  --**************************************************************

       waves: waveform(wpl);

  --**************************************************************
  -- processes that convert the WPL values to 1164 Logic Values
  --**************************************************************

  wav_stim_clk                        <= stim_1164(wpl.wpl( 1 ));
  wav_stim_load_bar                   <= stim_1164(wpl.wpl( 2 ));
  wav_stim_up_down_bar                <= stim_1164(wpl.wpl( 3 ));
  wav_stim_ent_bar                    <= stim_1164(wpl.wpl( 4 ));
  wav_stim_enp_bar                    <= stim_1164(wpl.wpl( 5 ));
  wav_stim_hgfedcba                   <= stim_1164(wpl.wpl( 6 TO 13 ));
  wav_expect_q                        <= expect_1164(wpl.wpl( 14 TO 21 ));
  wav_expect_rco_bar                  <= expect_1164(wpl.wpl( 22 ));

  --*****************************************
  -- UUT Port Map - Name Symantics Denote Usage
  --*****************************************

  u1: eight_bit_sync_ud_cntr
  PORT MAP(
    clk                               => wav_stim_clk,
    load_bar                          => wav_stim_load_bar,
    up_down_bar                       => wav_stim_up_down_bar,
    ent_bar                           => wav_stim_ent_bar,
    enp_bar                           => wav_stim_enp_bar,
    hgfedcba                          => wav_stim_hgfedcba,
    q                                 => actual_q,
    rco_bar                           => actual_rco_bar);
  --*********************************************************
  -- Monitor Processes To Verify The UUT Operational Response
  --*********************************************************

 monitor_q:
   PROCESS(actual_q, wav_expect_q)
   BEGIN
        ASSERT(compatible (actual => actual_q,
                           expected => wav_expect_q))
         REPORT "Error on Q output" SEVERITY WARNING;

   IF ( compatible ( actual_q,     wav_expect_q) ) THEN
     fail_signal <='L'; ELSE fail_signal <='1';
   END IF;
   END PROCESS;


 monitor_rco_bar:
   PROCESS(actual_rco_bar, wav_expect_rco_bar)
   BEGIN
        ASSERT(compatible (actual => actual_rco_bar,
                           expected => wav_expect_rco_bar))
         REPORT "Error on RCO_bar output" SEVERITY WARNING;

   IF ( compatible ( actual_rco_bar,     wav_expect_rco_bar) ) THEN
     fail_signal <='L'; ELSE fail_signal <='1';
   END IF;
   END PROCESS;


END eight_bit_sync_ud_cntr_test;
```

A6.6 External Vector File

```
% These are the test vectors to test the 8 Bit Up/Down Counter using WAVES.
% C L U E E HGFEDCBA QQQQQQQQ R
% L O p N N          HGFEDCBA C
% K A D T P                   O
%   D o b b                   b
%   b w a a                   a
%   a n r r                   r
%   r b
%     a
%     r

% Load zero.
  1 0 1 0 0 00000000 -------- - : 30 ns;
  1 1 1 0 0 -------- 00000000 1 : 30 ns;

% Count Up to 79.
  1 1 1 0 0 -------- 00000001 1 : 30 ns;
  1 1 1 0 0 -------- 00000010 1 : 30 ns;
  1 1 1 0 0 -------- 00000011 1 : 30 ns;
  1 1 1 0 0 -------- 00000100 1 : 30 ns;
  1 1 1 0 0 -------- 00000101 1 : 30 ns;
  1 1 1 0 0 -------- 00000110 1 : 30 ns;
  1 1 1 0 0 -------- 00000111 1 : 30 ns;
  1 1 1 0 0 -------- 00001000 1 : 30 ns;
  1 1 1 0 0 -------- 00001001 1 : 30 ns;
  1 1 1 0 0 -------- 00001010 1 : 30 ns;
  1 1 1 0 0 -------- 00001011 1 : 30 ns;
  1 1 1 0 0 -------- 00001100 1 : 30 ns;
  1 1 1 0 0 -------- 00001101 1 : 30 ns;
  1 1 1 0 0 -------- 00001110 1 : 30 ns;
  1 1 1 0 0 -------- 00001111 1 : 30 ns;
  1 1 1 0 0 -------- 00010000 1 : 30 ns;
  1 1 1 0 0 -------- 00010001 1 : 30 ns;
  1 1 1 0 0 -------- 00010010 1 : 30 ns;
  1 1 1 0 0 -------- 00010011 1 : 30 ns;
  1 1 1 0 0 -------- 00010100 1 : 30 ns;
  1 1 1 0 0 -------- 00010101 1 : 30 ns;
  1 1 1 0 0 -------- 00010110 1 : 30 ns;
  1 1 1 0 0 -------- 00010111 1 : 30 ns;
  1 1 1 0 0 -------- 00011000 1 : 30 ns;
  1 1 1 0 0 -------- 00011001 1 : 30 ns;
  1 1 1 0 0 -------- 00011010 1 : 30 ns;
  1 1 1 0 0 -------- 00011011 1 : 30 ns;
  1 1 1 0 0 -------- 00011100 1 : 30 ns;
  1 1 1 0 0 -------- 00011101 1 : 30 ns;
  1 1 1 0 0 -------- 00011110 1 : 30 ns;
  1 1 1 0 0 -------- 00011111 1 : 30 ns;
  1 1 1 0 0 -------- 00100000 1 : 30 ns;
  1 1 1 0 0 -------- 00100001 1 : 30 ns;
  1 1 1 0 0 -------- 00100010 1 : 30 ns;
  1 1 1 0 0 -------- 00100011 1 : 30 ns;
  1 1 1 0 0 -------- 00100100 1 : 30 ns;
  1 1 1 0 0 -------- 00100101 1 : 30 ns;
  1 1 1 0 0 -------- 00100110 1 : 30 ns;
  1 1 1 0 0 -------- 00100111 1 : 30 ns;
  1 1 1 0 0 -------- 00101000 1 : 30 ns;
  1 1 1 0 0 -------- 00101001 1 : 30 ns;
  1 1 1 0 0 -------- 00101010 1 : 30 ns;
  1 1 1 0 0 -------- 00101011 1 : 30 ns;
  1 1 1 0 0 -------- 00101100 1 : 30 ns;
  1 1 1 0 0 -------- 00101101 1 : 30 ns;
  1 1 1 0 0 -------- 00101110 1 : 30 ns;
  1 1 1 0 0 -------- 00101111 1 : 30 ns;
  1 1 1 0 0 -------- 00110000 1 : 30 ns;
  1 1 1 0 0 -------- 00110001 1 : 30 ns;
  1 1 1 0 0 -------- 00110010 1 : 30 ns;
  1 1 1 0 0 -------- 00110011 1 : 30 ns;
  1 1 1 0 0 -------- 00110100 1 : 30 ns;
  1 1 1 0 0 -------- 00110101 1 : 30 ns;
  1 1 1 0 0 -------- 00110110 1 : 30 ns;
  1 1 1 0 0 -------- 00110111 1 : 30 ns;
  1 1 1 0 0 -------- 00111000 1 : 30 ns;
  1 1 1 0 0 -------- 00111001 1 : 30 ns;
  1 1 1 0 0 -------- 00111010 1 : 30 ns;
  1 1 1 0 0 -------- 00111011 1 : 30 ns;
  1 1 1 0 0 -------- 00111100 1 : 30 ns;
  1 1 1 0 0 -------- 00111101 1 : 30 ns;
  1 1 1 0 0 -------- 00111110 1 : 30 ns;
  1 1 1 0 0 -------- 00111111 1 : 30 ns;
  1 1 1 0 0 -------- 01000000 1 : 30 ns;
  1 1 1 0 0 -------- 01000001 1 : 30 ns;
  1 1 1 0 0 -------- 01000010 1 : 30 ns;
  1 1 1 0 0 -------- 01000011 1 : 30 ns;
  1 1 1 0 0 -------- 01000100 1 : 30 ns;
  1 1 1 0 0 -------- 01000101 1 : 30 ns;
  1 1 1 0 0 -------- 01000110 1 : 30 ns;
  1 1 1 0 0 -------- 01000111 1 : 30 ns;
  1 1 1 0 0 -------- 01001000 1 : 30 ns;
  1 1 1 0 0 -------- 01001001 1 : 30 ns;
  1 1 1 0 0 -------- 01001010 1 : 30 ns;
  1 1 1 0 0 -------- 01001011 1 : 30 ns;
  1 1 1 0 0 -------- 01001100 1 : 30 ns;
  1 1 1 0 0 -------- 01001101 1 : 30 ns;
  1 1 1 0 0 -------- 01001110 1 : 30 ns;
  1 1 1 0 0 -------- 01001111 1 : 30 ns;

% Load 250.
  1 0 1 0 0 11111010 -------- - : 30 ns;

% Count Up from 250 to 16 to see if RCO_Bar goes low.
  1 1 1 0 0 -------- 11111010 1 : 30 ns;
  1 1 1 0 0 -------- 11111011 1 : 30 ns;
  1 1 1 0 0 -------- 11111100 1 : 30 ns;
  1 1 1 0 0 -------- 11111101 1 : 30 ns;
  1 1 1 0 0 -------- 11111110 1 : 30 ns;
  1 1 1 0 0 -------- 11111111 0 : 30 ns;
  1 1 1 0 0 -------- 00000000 1 : 30 ns;
  1 1 1 0 0 -------- 00000001 1 : 30 ns;
  1 1 1 0 0 -------- 00000010 1 : 30 ns;
  1 1 1 0 0 -------- 00000011 1 : 30 ns;
  1 1 1 0 0 -------- 00000100 1 : 30 ns;
  1 1 1 0 0 -------- 00000101 1 : 30 ns;
  1 1 1 0 0 -------- 00000110 1 : 30 ns;
  1 1 1 0 0 -------- 00000111 1 : 30 ns;
  1 1 1 0 0 -------- 00001000 1 : 30 ns;
  1 1 1 0 0 -------- 00001001 1 : 30 ns;
  1 1 1 0 0 -------- 00001010 1 : 30 ns;
  1 1 1 0 0 -------- 00001011 1 : 30 ns;
  1 1 1 0 0 -------- 00001100 1 : 30 ns;
  1 1 1 0 0 -------- 00001101 1 : 30 ns;
  1 1 1 0 0 -------- 00001110 1 : 30 ns;
  1 1 1 0 0 -------- 00001111 1 : 30 ns;
  1 1 1 0 0 -------- 00010000 1 : 30 ns;

% Load 100.
  1 0 0 0 0 01100100 -------- - : 30 ns;

% Count Down through 0 to 245 to see if RCO_Bar goes low.
  1 1 0 0 0 -------- 01100100 1 : 30 ns;
  1 1 0 0 0 -------- 01100011 1 : 30 ns;
  1 1 0 0 0 -------- 01100010 1 : 30 ns;
  1 1 0 0 0 -------- 01100001 1 : 30 ns;
  1 1 0 0 0 -------- 01100000 1 : 30 ns;
  1 1 0 0 0 -------- 01011111 1 : 30 ns;
  1 1 0 0 0 -------- 01011110 1 : 30 ns;
  1 1 0 0 0 -------- 01011101 1 : 30 ns;
  1 1 0 0 0 -------- 01011100 1 : 30 ns;
  1 1 0 0 0 -------- 01011011 1 : 30 ns;
  1 1 0 0 0 -------- 01011010 1 : 30 ns;
  1 1 0 0 0 -------- 01011001 1 : 30 ns;
  1 1 0 0 0 -------- 01011000 1 : 30 ns;
  1 1 0 0 0 -------- 01010111 1 : 30 ns;
  1 1 0 0 0 -------- 01010110 1 : 30 ns;
  1 1 0 0 0 -------- 01010101 1 : 30 ns;
  1 1 0 0 0 -------- 01010100 1 : 30 ns;
  1 1 0 0 0 -------- 01010011 1 : 30 ns;
  1 1 0 0 0 -------- 01010010 1 : 30 ns;
  1 1 0 0 0 -------- 01010001 1 : 30 ns;
  1 1 0 0 0 -------- 01010000 1 : 30 ns;
  1 1 0 0 0 -------- 01001111 1 : 30 ns;
  1 1 0 0 0 -------- 01001110 1 : 30 ns;
  1 1 0 0 0 -------- 01001101 1 : 30 ns;
  1 1 0 0 0 -------- 01001100 1 : 30 ns;
```

```
1 1 0 0 0 -------- 01001011 1 : 30 ns;
1 1 0 0 0 -------- 01001010 1 : 30 ns;
1 1 0 0 0 -------- 01001001 1 : 30 ns;
1 1 0 0 0 -------- 01001000 1 : 30 ns;
1 1 0 0 0 -------- 01000111 1 : 30 ns;
1 1 0 0 0 -------- 01000110 1 : 30 ns;
1 1 0 0 0 -------- 01000101 1 : 30 ns;
1 1 0 0 0 -------- 01000100 1 : 30 ns;
1 1 0 0 0 -------- 01000011 1 : 30 ns;
1 1 0 0 0 -------- 01000010 1 : 30 ns;
1 1 0 0 0 -------- 01000001 1 : 30 ns;
1 1 0 0 0 -------- 01000000 1 : 30 ns;
1 1 0 0 0 -------- 00111111 1 : 30 ns;
1 1 0 0 0 -------- 00111110 1 : 30 ns;
1 1 0 0 0 -------- 00111101 1 : 30 ns;
1 1 0 0 0 -------- 00111100 1 : 30 ns;
1 1 0 0 0 -------- 00111011 1 : 30 ns;
1 1 0 0 0 -------- 00111010 1 : 30 ns;
1 1 0 0 0 -------- 00111001 1 : 30 ns;
1 1 0 0 0 -------- 00111000 1 : 30 ns;
1 1 0 0 0 -------- 00110111 1 : 30 ns;
1 1 0 0 0 -------- 00110110 1 : 30 ns;
1 1 0 0 0 -------- 00110101 1 : 30 ns;
1 1 0 0 0 -------- 00110100 1 : 30 ns;
1 1 0 0 0 -------- 00110011 1 : 30 ns;
1 1 0 0 0 -------- 00110010 1 : 30 ns;
1 1 0 0 0 -------- 00110001 1 : 30 ns;
1 1 0 0 0 -------- 00110000 1 : 30 ns;
1 1 0 0 0 -------- 00101111 1 : 30 ns;
1 1 0 0 0 -------- 00101110 1 : 30 ns;
1 1 0 0 0 -------- 00101101 1 : 30 ns;
1 1 0 0 0 -------- 00101100 1 : 30 ns;
1 1 0 0 0 -------- 00101011 1 : 30 ns;
1 1 0 0 0 -------- 00101010 1 : 30 ns;
1 1 0 0 0 -------- 00101001 1 : 30 ns;
1 1 0 0 0 -------- 00101000 1 : 30 ns;
1 1 0 0 0 -------- 00100111 1 : 30 ns;
1 1 0 0 0 -------- 00100110 1 : 30 ns;
1 1 0 0 0 -------- 00100101 1 : 30 ns;
1 1 0 0 0 -------- 00100100 1 : 30 ns;
1 1 0 0 0 -------- 00100011 1 : 30 ns;
1 1 0 0 0 -------- 00100010 1 : 30 ns;
1 1 0 0 0 -------- 00100001 1 : 30 ns;
1 1 0 0 0 -------- 00100000 1 : 30 ns;
1 1 0 0 0 -------- 00011111 1 : 30 ns;
1 1 0 0 0 -------- 00011110 1 : 30 ns;
1 1 0 0 0 -------- 00011101 1 : 30 ns;
1 1 0 0 0 -------- 00011100 1 : 30 ns;
1 1 0 0 0 -------- 00011011 1 : 30 ns;
1 1 0 0 0 -------- 00011010 1 : 30 ns;
1 1 0 0 0 -------- 00011001 1 : 30 ns;
1 1 0 0 0 -------- 00011000 1 : 30 ns;
1 1 0 0 0 -------- 00010111 1 : 30 ns;
1 1 0 0 0 -------- 00010110 1 : 30 ns;
1 1 0 0 0 -------- 00010101 1 : 30 ns;
1 1 0 0 0 -------- 00010100 1 : 30 ns;
1 1 0 0 0 -------- 00010011 1 : 30 ns;
1 1 0 0 0 -------- 00010010 1 : 30 ns;
1 1 0 0 0 -------- 00010001 1 : 30 ns;
1 1 0 0 0 -------- 00010000 1 : 30 ns;
1 1 0 0 0 -------- 00001111 1 : 30 ns;
1 1 0 0 0 -------- 00001110 1 : 30 ns;
1 1 0 0 0 -------- 00001101 1 : 30 ns;
1 1 0 0 0 -------- 00001100 1 : 30 ns;
1 1 0 0 0 -------- 00001011 1 : 30 ns;
1 1 0 0 0 -------- 00001010 1 : 30 ns;
1 1 0 0 0 -------- 00001001 1 : 30 ns;
1 1 0 0 0 -------- 00001000 1 : 30 ns;
1 1 0 0 0 -------- 00000111 1 : 30 ns;
1 1 0 0 0 -------- 00000110 1 : 30 ns;
1 1 0 0 0 -------- 00000101 1 : 30 ns;
1 1 0 0 0 -------- 00000100 1 : 30 ns;
1 1 0 0 0 -------- 00000011 1 : 30 ns;
1 1 0 0 0 -------- 00000010 1 : 30 ns;
1 1 0 0 0 -------- 00000001 1 : 30 ns;
1 1 0 0 0 -------- 00000000 0 : 30 ns;
1 1 0 0 0 -------- 11111111 1 : 30 ns;
1 1 0 0 0 -------- 11111110 1 : 30 ns;
1 1 0 0 0 -------- 11111101 1 : 30 ns;
1 1 0 0 0 -------- 11111100 1 : 30 ns;
1 1 0 0 0 -------- 11111011 1 : 30 ns;
```

```
  1 1 0 0 0 -------- 11111010 1 : 30 ns;
  1 1 0 0 0 -------- 11111001 1 : 30 ns;
  1 1 0 0 0 -------- 11111000 1 : 30 ns;
  1 1 0 0 0 -------- 11110111 1 : 30 ns;
  1 1 0 0 0 -------- 11110110 1 : 30 ns;
  1 1 0 0 0 -------- 11110101 1 : 30 ns;

% Load zero, count up to 2, disable counting with ENT_Bar and ENP_Bar
% for two clocks then count down past 0 to 253.
  1 0 1 0 0 00000000 -------- - : 30 ns;
  1 1 1 0 0 -------- 00000000 1 : 30 ns;
  1 1 1 1 1 -------- 00000001 1 : 30 ns;
  1 1 1 1 1 -------- 00000010 1 : 30 ns;
  1 1 0 0 0 -------- 00000010 1 : 30 ns;
  1 1 0 0 0 -------- 00000010 1 : 30 ns;
  1 1 0 0 0 -------- 00000001 1 : 30 ns;
  1 1 0 0 0 -------- 00000000 0 : 30 ns;
  1 1 0 0 0 -------- 11111111 1 : 30 ns;
  1 1 0 0 0 -------- 11111110 1 : 30 ns;
  1 1 0 0 0 -------- 11111101 1 : 30 ns;

% Load zero, count up to 2, disable counting with only ENT_Bar high
% for two clocks then count down past 0 to 253.
  1 0 1 0 0 00000000 -------- - : 30 ns;
  1 1 1 0 0 -------- 00000000 1 : 30 ns;
  1 1 1 1 0 -------- 00000001 1 : 30 ns;
  1 1 1 1 0 -------- 00000010 1 : 30 ns;
  1 1 0 0 0 -------- 00000010 1 : 30 ns;
  1 1 0 0 0 -------- 00000010 1 : 30 ns;
  1 1 0 0 0 -------- 00000001 1 : 30 ns;
  1 1 0 0 0 -------- 00000000 0 : 30 ns;
  1 1 0 0 0 -------- 11111111 1 : 30 ns;
  1 1 0 0 0 -------- 11111110 1 : 30 ns;
  1 1 0 0 0 -------- 11111101 1 : 30 ns;

% Load zero, count up to 2, disable counting with only ENP_Bar high
% for two clocks then count down past 0 to 253.
  1 0 1 0 0 00000000 -------- - : 30 ns;
  1 1 1 0 0 -------- 00000000 1 : 30 ns;
  1 1 1 0 1 -------- 00000001 1 : 30 ns;
  1 1 1 0 1 -------- 00000010 1 : 30 ns;
  1 1 0 0 0 -------- 00000010 1 : 30 ns;
  1 1 0 0 0 -------- 00000010 1 : 30 ns;
  1 1 0 0 0 -------- 00000001 1 : 30 ns;
  1 1 0 0 0 -------- 00000000 0 : 30 ns;
  1 1 0 0 0 -------- 11111111 1 : 30 ns;
  1 1 0 0 0 -------- 11111110 1 : 30 ns;
  1 1 0 0 0 -------- 11111101 1 : 30 ns;
```

APPENDIX 7: 54LS/74LS299 8 - INPUT UNIVERSAL SHIFT/STORAGE REGISTER

A7.1 VHDL Model

```
-- TITLE: FAIRCHILD 8-input universal shift/storage register; 54LS/4LS299
-- DATE : 16 June 1995
--
-- VERSION : 2.0
-- FILENAME : register_with_timing.vhd
-- FUNCTION : Entity and architecture for 54LS/74LS299; '299 is 8-bit universal
-- shift/storage
-- register with 3-state outputs. Four modes of operation are possible: hold(store)
-- ,shift left, shift right and load data. The parallel load inputs and -- flip-flop
-- outputs are multiplexed to reduce total number of package pins. Separate outputs
-- are provided for flip-flops Q0 and Q7 to allow easy cascading. A separate active
-- low master reset is used to reset te register.
--
-- Dataflow Model.
--
-- AUTHOR       : James M. Nagy
-- ORGANIZATION: Rome Laboratory
--
-- PURPOSE AND USE: This was written as an bi-directional example to help build the
-- Rome Laboratory WAVES tools.
--
-- TIMING: Max
--
-- DEVELOPMENT PLATFORM : Sun Sparc Station IPX
-- VHDL SOFTWARE VERSION: Synopsys 3.3a
--
-- HISTORY:
-- 28 Apr 95 - v1.0 -  Initial version, functional only, no timing.
-- 31 May 95 - v2.0 -  Final version with worst case timing.
--


LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY shift_register IS
GENERIC( to_min : TIME := 2 ns; clk_to_io : TIME := 32 ns;
         clk_to_q0_or_7 : TIME := 34 ns; reset_to_outputs : TIME := 36 ns;
         out_enable_z_to_hi : TIME := 25 ns; out_enable_hi_to_z : TIME := 27 ns;
         out_enable_z_to_lo : TIME := 30 ns; out_enable_lo_to_z : TIME := 34 ns;
         setup_data_to_clk : TIME := 30 ns; setup_selection_to_clk : TIME := 41 ns;
         hold_all_inputs_to_clk : TIME := 0 ns; reset_to_clock_hi : TIME := 5 ns;
         reset_lo_pulse_width : TIME := 22 ns; clk_hi_pulse_width : TIME := 30 ns);

PORT( selection, enable_out : IN STD_LOGIC_VECTOR(0 TO 1);
      clock,data_0,data_7,master_reset : IN STD_LOGIC;
      io : INOUT STD_LOGIC_VECTOR(0 TO 7) BUS;
      out_0,out_7 : OUT STD_LOGIC);

BEGIN

-- Master reset width check
     ASSERT
       (NOT ((master_reset = '0' AND NOT master_reset'STABLE) AND
        master_reset'LAST_EVENT >= reset_lo_pulse_width))
-- NOT Master_Reset'DELAYED'STABLE(Reset_lo_pulse_width)))
     REPORT
       "RESET PULSE WIDTH TOO SHORT"
     SEVERITY WARNING;
```

```
-- Minimum clock width pulse check
      ASSERT
         (NOT ((clock='1' AND NOT clock'STABLE) AND
           clock'LAST_EVENT >= clk_hi_pulse_width))
-- NOT Clock'delayed'stable(clk_hi_pulse_width)))
      REPORT
         "CLOCK PULSE WIDTH TOO SHORT"
       SEVERITY WARNING;

-- Setup timing checks for selection, IO, Data_0 and Data_7
      ASSERT
         (NOT (clock = '1' AND clock'EVENT AND NOT
                selection'STABLE(setup_selection_to_clk)))
      REPORT
         "Setup TIME VIOLATION ON selection PINS"
       SEVERITY WARNING;

      ASSERT
         (NOT (clock = '1' AND clock'EVENT AND NOT io'STABLE(setup_data_to_clk) AND
            selection ="11"))
      REPORT
         "Setup TIME VIOLATION ON IO PINS"
       SEVERITY WARNING;

      ASSERT
         (NOT (clock = '1' AND clock'EVENT AND NOT data_0'STABLE(setup_data_to_clk)))
      REPORT
         "Setup TIME VIOLATION ON Data_0 PIN"
       SEVERITY WARNING;

      ASSERT
         (NOT (clock = '1' AND clock'EVENT AND NOT data_7'STABLE(setup_data_to_clk)))
      REPORT
         "Setup TIME VIOLATION ON Data_7 PIN"
       SEVERITY WARNING;

-- Hold timing checks for IO,
       ASSERT
         (NOT (clock = '1' AND io'EVENT AND NOT clock'STABLE(hold_all_inputs_to_clk)))
      REPORT
         "HOLD TIME VIOLATION ON IO PINS"
       SEVERITY WARNING;

     ASSERT
         (NOT (clock = '1' AND selection'EVENT AND NOT
                clock'STABLE(hold_all_inputs_to_clk)))
      REPORT
         "HOLD TIME VIOLATION ON selection PINS"
       SEVERITY WARNING;

      ASSERT
         (NOT (clock = '1' AND data_0'EVENT AND NOT
                clock'STABLE(hold_all_inputs_to_clk)))
      REPORT
         "HOLD TIME VIOLATION ON Data_0 PIN"
       SEVERITY WARNING;

      ASSERT
         (NOT (clock = '1' AND data_7'EVENT AND NOT
                clock'STABLE(hold_all_inputs_to_clk)))
      REPORT
         "HOLD TIME VIOLATION ON Data_7 PIN"
       SEVERITY WARNING;

END shift_register;
```

```
ARCHITECTURE behavioral OF  shift_register IS
  SIGNAL q :  STD_LOGIC_VECTOR(0 TO 7) REGISTER;

BEGIN

 memory_reset: BLOCK ( master_reset = '0')
 BEGIN
     q <=  GUARDED "00000000";
 END BLOCK;

 clocking : BLOCK (clock = '1' AND NOT clock'STABLE AND master_reset = '1')
   BEGIN
     shift_right: BLOCK(selection = "10" AND GUARD)
       BEGIN
         q <= GUARDED data_0 & q(0 TO 6);
       END BLOCK shift_right;
--
     shift_left: BLOCK(selection = "01" AND GUARD)
       BEGIN
         q<= GUARDED q(1 TO 7) & data_7;
       END BLOCK shift_left;
--
    parrel_load : BLOCK(selection = "11" AND GUARD)
      BEGIN
         q<= GUARDED io;
      END BLOCK parrel_load;

--      Hold : BLOCK(selection = "00" AND GUARD)
--        BEGIN
--          Q <= GUARDED Q;
--        END BLOCK Hold;

  END BLOCK clocking;

  check_oe  : BLOCK ( enable_out = "00"   )
   BEGIN
      io <= GUARDED q AFTER reset_to_outputs WHEN q = "00000000" ELSE
                    q AFTER out_enable_z_to_hi WHEN io = "ZZZZZZZZ" ELSE
                    q AFTER clk_to_io;
   END BLOCK check_oe;

  check_z  : BLOCK( TRUE )
   BEGIN
     io(0)  <= GUARDED 'Z' AFTER out_enable_hi_to_z WHEN io(0)  =  '1'  ELSE
                      'Z' AFTER out_enable_lo_to_z;
     io(1)  <= GUARDED 'Z' AFTER out_enable_hi_to_z WHEN io(0)  =  '1'  ELSE
                      'Z' AFTER out_enable_lo_to_z;
     io(2)  <= GUARDED 'Z' AFTER out_enable_hi_to_z WHEN io(0)  =  '1'  ELSE
                      'Z' AFTER out_enable_lo_to_z;
     io(3)  <= GUARDED 'Z' AFTER out_enable_hi_to_z WHEN io(0)  =  '1'  ELSE
                      'Z' AFTER out_enable_lo_to_z;
     io(4)  <= GUARDED 'Z' AFTER out_enable_hi_to_z WHEN io(0)  =  '1'  ELSE
                      'Z' AFTER out_enable_lo_to_z;
     io(5)  <= GUARDED 'Z' AFTER out_enable_hi_to_z WHEN io(0)  =  '1'  ELSE
                      'Z' AFTER out_enable_lo_to_z;
     io(6)  <= GUARDED 'Z' AFTER out_enable_hi_to_z WHEN io(0)  =  '1'  ELSE
                      'Z' AFTER out_enable_lo_to_z;
     io(7)  <= GUARDED 'Z' AFTER out_enable_hi_to_z WHEN io(0)  =  '1'  ELSE
                      'Z' AFTER out_enable_lo_to_z;
   END BLOCK check_z;

   out_0  <= q(0)  AFTER clk_to_q0_or_7;
   out_7  <= q(7)  AFTER clk_to_q0_or_7;

END behavioral;
```

A7.2 WAVES HEADER FILE

```
-- **************************************************
--
-- ******** Header File for Entity: shift_register
--
-- **************************************************
-- **************************************************
--
-- Data Set Identification Information
--
TITLE          A General Description
DEVICE_ID      shift_register

DATE           Fri Jul 28 10:44:44 1995
ORIGIN         Company X Design Team
AUTHOR         Company or Person
AUTHOR         Maybe Multiple ... Companies or People
DATE           Fri Jul 28 10:44:44 1995
ORIGIN         Modified by Company X Design Team
AUTHOR         Who did it Company or Person

OTHER          Use TestBench register_tstbench.vhd
OTHER          Any general comments you want
OTHER          Built Using the WAVES-VHDL 1164 STD Libraries
--
-- Data Set Construction Information
--
waves_filename   register_pins.vhd         WORK
LIBRARY          waves_1164;
USE              waves_1164.waves_1164_pin_codes.ALL;
USE              waves_1164.waves_1164_logic_value.ALL;
USE              waves_1164.waves_interface.ALL;
USE              WORK.uut_test_pins.ALL;
waves_unit       waves_objects                         WORK
waves_filename   register_wgen.vhd         WORK
--
external_filename   vectors.txt          vectors
--
waveform_generator_procedure     WORK.waves_shift_register.waveform
```

A7.3 WAVES PINS PACKAGE

```
-- ******** This File Was Automatically Generated  ********
-- ******** By The WAVES-VHDL TestBench Tool        ********
-- ******** Generated for Entity: shift_register
-- ******** This File Was Generated on: Fri Jul 28 10:44:44 1995
--
--
PACKAGE uut_test_pins IS
TYPE test_pins IS (selection_0, selection_1, enable_out_0, enable_out_1,
          clock, data_0, data_7, master_reset, io_0, io_1, io_2, io_3, io_4,
          io_5, io_6, io_7, out_0, out_7);
END uut_test_pins;
```

A7.4 WAVES GENERATOR PACKAGE

```
-- ******** This File Was Automatically Generated   ********
-- ******** By The WAVES-VHDL TestBench Tool        ********
-- ******** Generated for Entity: shift_register
-- ******** This File Was Generated on: Wed Jun  7 17:06:15 1995
--
--

LIBRARY waves_std;
USE waves_std.waves_standard.ALL;
LIBRARY waves_1164;
USE STD.TEXTIO.ALL;
USE waves_1164.waves_1164_frames.ALL;
USE waves_1164.waves_1164_pin_codes.ALL;
USE waves_1164.waves_interface.ALL;
USE WORK.waves_objects.ALL;
USE WORK.uut_test_pins.ALL;

PACKAGE wgp_shift_register IS
     PROCEDURE  waveform(SIGNAL wpl : INOUT waves_port_list);
END wgp_shift_register;

-------------------------------------------------

PACKAGE BODY wgp_shift_register IS

-- This is the uut pin declaration pin and ordering
-- Remember you need to match the External file to This order
--
--selection_0, selection_1, enable_out_0, enable_out_1, clock, data_0,
-- data_7, master_reset, io_0, io_1, io_2, io_3, io_4, io_5, io_6, io_7,
-- out_0, out_7

    PROCEDURE  waveform(SIGNAL wpl : INOUT waves_port_list) IS

       FILE vector_file : TEXT IS IN "vectors.txt";

       VARIABLE vector : file_slice := new_file_slice;

        -- declare time constants to use or use time literals
        -- constants or time literals can be used as the frame time values

         CONSTANT selection: pinset:= new_pinset(( selection_0, selection_1));

         CONSTANT enable_out: pinset:= new_pinset(( enable_out_0, enable_out_1));

         CONSTANT io: pinset:= new_pinset(( io_0, io_1, io_2, io_3, io_4,
                      io_5, io_6, io_7));

         CONSTANT outputs: pinset:= new_pinset((out_0, out_7));

         CONSTANT in_pins: pinset:= new_pinset((data_0, data_7,
                              master_reset));

         CONSTANT inputs: pinset:= in_pins OR selection OR
                      enable_out;
```

```
-- You Must Modify the Frame Sets, Cycle Times, Frame Times, Groupings
-- This file Is only a Template to get you started
--
        --
        -- Declare The Frame Sets (timing sets)
        --
        VARIABLE wtl : wave_timing_list (1 TO 2) := (
        --
        -- Frame Set 1
        --
        (  delay  => delay ( 100 ns ),
           timing => new_time_data(
            new_frame_set_array(pulse_high( 50 ns, 80 ns), clock) +
            new_frame_set_array(window( 85 ns, 95 ns), io) +
-- or drive format           New_frame_set_array(Non_return( ns), IO) +
            new_frame_set_array(non_return( 5 ns), inputs) +
            new_frame_set_array(window( 85 ns, 95 ns), outputs)
         )),
        --
        -- Frame Set 2
        --
        (  delay  => delay ( 100 ns ),
           timing => new_time_data(
            new_frame_set_array(pulse_high( 50 ns, 80 ns), clock) +
-- select compare            New_frame_set_array(Window( ns, ns), IO) +
            new_frame_set_array(non_return( 5 ns), io) +
            new_frame_set_array(non_return( 5 ns), inputs) +
            new_frame_set_array(window( 85 ns, 95 ns), outputs)
         ) ) );

      BEGIN
        LOOP
          read_file_slice (vector_file, vector);  -- get first vector
          EXIT WHEN vector.end_of_file;
          apply(wpl, vector.codes.ALL, wtl(vector.fs_integer));
        END LOOP;

      END waveform;

END wgp_shift_register;
```

A7.5 WAVES TESTBENCH CODE

```
-- ********.This File Was Automatically Generated    ********
-- ******** By The WAVES-VHDL TestBench Tool         ********
-- ******** Generated for Entity: shift_register
-- ******** This File Was Generated on: Fri Jul 28 10:44:44 1995
--
--
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

LIBRARY waves_1164;
USE waves_1164.waves_1164_utilities.ALL;

USE WORK.uut_test_pins.ALL;
USE WORK.waves_objects.ALL;

USE WORK.wgp_shift_register.ALL;
-- Include component libary references here
-- User Must Modify And ADD component libary references here
-- Include component libary references here

ENTITY test_bench IS
END test_bench;


ARCHITECTURE shift_register_test OF test_bench IS


   --*******************************************************
   --***********CONFIGURATION SPECIFICATION ***************
   --*******************************************************

   COMPONENT shift_register
    PORT ( selection              :IN    STD_LOGIC_VECTOR(  0 TO  1 );
           enable_out             :IN    STD_LOGIC_VECTOR(  0 TO  1 );
           clock                  :IN    STD_LOGIC;
           data_0                 :IN    STD_LOGIC;
           data_7                 :IN    STD_LOGIC;
           master_reset           :IN    STD_LOGIC;
           io                     :INOUT STD_LOGIC_VECTOR(  0 TO  7 );
           out_0                  :OUT   STD_LOGIC;
           out_7                  :OUT   STD_LOGIC);
     END COMPONENT;

 -- Modify entity use statement
 -- User Must Modify modify and declare correct
 --   .. Architecture, Library, Component ..
 -- Modify entity use statement
FOR ALL:shift_register USE ENTITY WORK.shift_register(behavioral);

   --***********************************************************
  -- stimulus signals for the waveforms mapped into UUT INPUTS
   --***********************************************************

    SIGNAL wav_stim_selection        :STD_LOGIC_VECTOR(  0 TO  1 );
    SIGNAL wav_stim_enable_out       :STD_LOGIC_VECTOR(  0 TO  1 );
    SIGNAL wav_stim_clock            :STD_LOGIC;
    SIGNAL wav_stim_data_0           :STD_LOGIC;
    SIGNAL wav_stim_data_7           :STD_LOGIC;
    SIGNAL wav_stim_master_reset     :STD_LOGIC;

   --***************************************************
  -- Expected signals used in monitoring the UUT OUTPUTS
   --***************************************************

    SIGNAL fail_signal               :STD_LOGIC;
    SIGNAL wav_expect_io             :STD_ULOGIC_VECTOR(  0 TO  7 );
    SIGNAL wav_expect_out_0          :STD_LOGIC;
    SIGNAL wav_expect_out_7          :STD_LOGIC;
  --*****************************************************
  -- UUT Output signals used In Monitoring ACTUAL Values
  --*****************************************************

   SIGNAL actual_out_0               :STD_LOGIC;
   SIGNAL actual_out_7               :STD_LOGIC;

  --*********************************************************
  -- Bi_directional signals used  for stimulus signals mapped
  -- into UUT INPUTS and also monitoring the UUT OUTPUTS
  --*********************************************************

   SIGNAL bi_direc_io               :STD_LOGIC_VECTOR(  0 TO  7 );

  --***************************************************************
  -- WAVES signals OUTPUTing each slice of the waves port list
  --***************************************************************

       SIGNAL wpl  : waves_port_list;

BEGIN
  --
  --***************************************************************
  -- process that generates the WAVES waveform
  --***************************************************************

      waves: waveform(wpl);

  --***************************************************************
  -- processes that convert the WPL values to 1164 Logic Values
  --***************************************************************

  wav_stim_selection                  <= stim_1164(wpl.wpl( 1 TO 2 ));
  wav_stim_enable_out                 <= stim_1164(wpl.wpl( 3 TO 4 ));
  wav_stim_clock                      <= stim_1164(wpl.wpl( 5 ));
  wav_stim_data_0                     <= stim_1164(wpl.wpl( 6 ));
  wav_stim_data_7                     <= stim_1164(wpl.wpl( 7 ));
  wav_stim_master_reset               <= stim_1164(wpl.wpl( 8 ));
  bi_direc_io                         <= bi_dir_1164(wpl.wpl( 9 to 16 ));
  wav_expect_io                       <= expect_1164(wpl.wpl( 9 TO 16 ));
  wav_expect_out_0                    <= expect_1164(wpl.wpl( 17 ));
  wav_expect_out_7                    <= expect_1164(wpl.wpl( 18 ));

  --******************************************
  -- UUT Port Map - Name Symantics Denote Usage
  --******************************************

  u1: shift_register
  PORT MAP(
    selection                         => wav_stim_selection,
    enable_out                        => wav_stim_enable_out,
    clock                             => wav_stim_clock,
    data_0                            => wav_stim_data_0,
    data_7                            => wav_stim_data_7,
    master_reset                      => wav_stim_master_reset,
    io                                => bi_direc_io,
    out_0                             => actual_out_0,
    out_7                             => actual_out_7);
   --*********************************************************
   -- Monitor Processes To Verify The UUT Operational Response
   --*********************************************************

 monitor_io:
   PROCESS(bi_direc_io, wav_expect_io)
   BEGIN
        ASSERT(compatible (actual => bi_direc_io,
                           expected => wav_expect_io))
         REPORT "Error on IO output" SEVERITY WARNING;

   IF ( compatible ( bi_direc_io,    wav_expect_io) ) THEN
     fail_signal <='L'; ELSE fail_signal <='1';
   END IF;
   END PROCESS;


 monitor_out_0:
   PROCESS(actual_out_0, wav_expect_out_0)
   BEGIN
        ASSERT(compatible (actual => actual_out_0,
                           expected => wav_expect_out_0))
         REPORT "Error on OUT_0 output" SEVERITY WARNING;

   IF ( compatible ( actual_out_0,    wav_expect_out_0) ) THEN
     fail_signal <='L'; ELSE fail_signal <='1';
   END IF;
   END PROCESS;


 monitor_out_7:
   PROCESS(actual_out_7, wav_expect_out_7)
   BEGIN
        ASSERT(compatible (actual => actual_out_7,
                           expected => wav_expect_out_7))
         REPORT "Error on OUT_7 output" SEVERITY WARNING;

   IF ( compatible ( actual_out_7,    wav_expect_out_7) ) THEN
     fail_signal <='L'; ELSE fail_signal <='1';
   END IF;
   END PROCESS;

END shift_register_test;
```

A7.6 External Vector File

```
%  01          07  t                   t
%  11   12 k   tt  s    01234567       s
%  ee   ee l   aa  r    oooooooo  07   e
%  ss   oo c   dd  m    iiiiiiii  qq   t

% clear
   --   00 -   --  0    00000000  00 : 1 ;

%shift  right
   10   00 1   11  1    10000000  10 : 1 ;
   10   00 1   11  1    11000000  10 : 1 ;
   10   00 1   11  1    11100000  10 : 1 ;
   10   00 1   11  1    11110000  10 : 1 ;
   10   00 1   11  1    11111000  10 : 1 ;
   10   00 1   11  1    11111100  10 : 1 ;

%shift  left
   01   00 1   11  1    11111001  11 : 1 ;
   01   00 1   11  1    11110011  11 : 1 ;
   01   00 1   11  1    11100111  11 : 1 ;
   01   00 1   11  1    11001111  11 : 1 ;
   01   00 1   10  1    10011110  10 : 1 ;

% hold
   00   00 1   10  1    10011110  10 : 1 ;
   00   00 1   10  1    10011110  10 : 1 ;
   00   00 1   10  1    10011110  10 : 1 ;
   00   00 1   10  1    10011110  10 : 1 ;

% load
   11   10 1   10  1    01010101  01 : 2 ;

% enable & shift
   01   01 1   01  1    ZZZZZZZZ  11 : 1 ;
   01   00 1   01  1    01010111  01 : 1 ;
```